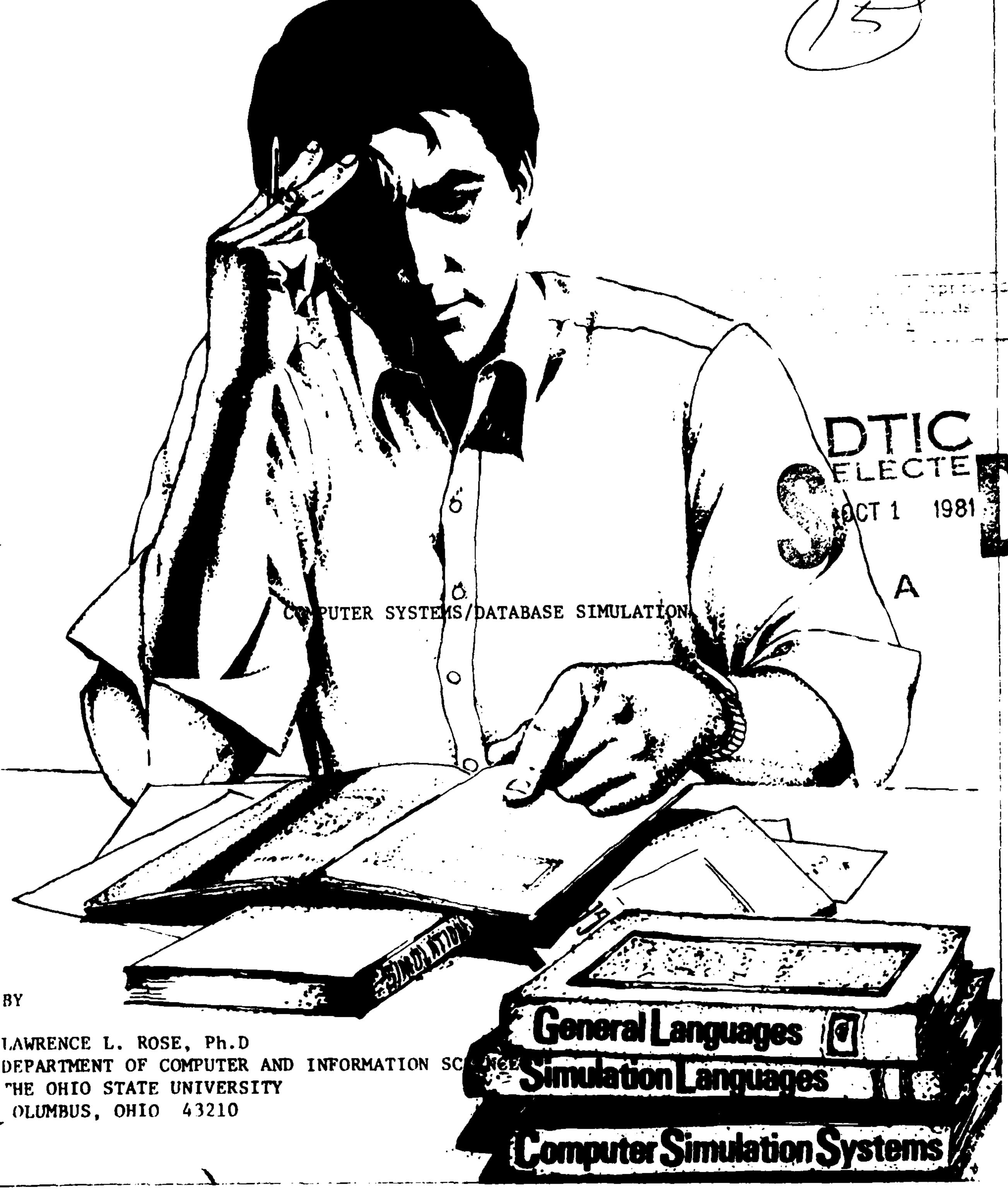
AD-A104 930 OHIO STATE UNIV COLUMBUS DEPT OF COMPUTER AND INFORM--ETC F/6 9/2
COMPUTER SYSTEMS/DATABASE SIMULATION.(U)
OCT 78 L L ROSE

UNCLASSIFIED NL

1 OF 2
AD A
104 930

A FINAL REPORT TO THE ARMY INSTITUTE FOR RESEARCH IN MANAGEMENT INFORMATION AND COMPUTER SCIENCE (AIRMICS)

15



COMPUTER SYSTEMS/DATABASE SIMULATION

BY

LAWRENCE L. ROSE, Ph.D
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE
THE OHIO STATE UNIVERSITY
COLUMBUS, OHIO 43210

AIRMICS FINAL REPORT*

Computer Systems/Database Simulation

by

Lawrence L. Rose, Ph.D
Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210

(12)

15 Oct~~ober 1~~978

A

*This research was performed for AIRMICS through the United States Army Research Office with the financial support of the Battelle Scientific Services Program - D.O. 0909.

[illegible]

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

ABSTRACT

This report evaluates the database modeling capabilities of three large scale computer simulation facilities. The systems overviewed and contrasted are: 1) CASE, a TESDATA commercial product; 2) DIMUI, a TECHNION University research product; and 3) IPSS, an Ohio State University research product. Chapters are further provided which overview simulation techniques and the area of database management systems.

Criteria are derived to form the basis for evaluation of the three subject systems. Upon consolidation into a linear objective function, the systems are rated: IPSS, DIMUI, and lastly CASE. System deficiencies are revealed and fruitful areas for further research are objectively discussed.

1. REPORT SYNOPSIS

This report is in response to the Laboratory Research Cooperative Program (LRCP) Statement of Work (TCN: 78-276) which required the services of a Computer Scientist on a short-term basis to provide a consolidation of the Army Institute for Research in Management Information and Computer Science (AIRMICS) R&D effort in the development of computer simulation capabilities. The object of this research effort was to produce logical coherent sets of documentation on three current simulation projects showing their interrelationship and complementary nature.

Specific tasks were defined as follows:

1. Gather and analyze all reports and system documentation concerning the DIMUI, CASE/IDMS, and IPSS simulators.
2. Prepare a report clearly reflecting the interrelationship of the above methodologies.
3. Utilizing the FEDSIM Report "Evaluation of DBMS Modeling Approaches", prepare recommendations for a simulation research program.
4. Outline a program of instruction necessary for training of computer specialists in computer simulation.

The contents of this Final Report to AIRMICS document the results of a 90 day study of CASE, IPSS, and DIMUI by the author. Tasks 1, 2, and 3 have been completed in full; task 4 was not entertained due to the lack of time to consider it at an appropriate level of detail.

This report provides more than was required by the SOW task descriptions: it also includes insights, background, and motivation for database simulation modeling. Chapter 2 provides a suitable background for simulation modeling;

motivating factors for simulation include cost, technology, morality, time, risk, knowledge, and applicability. The appropriateness of general programming languages, general simulation languages, and computer simulation systems to computer database simulation is discussed.

Chapter 3 builds a background for the existance of database management systems given the factors: integration, data independence, security and privacy, integrity, etc. The three basic models - relational, hierarchical, network - are defined and amplified with an example that carries through all three models. Given this background in simulation and DBMS, the reader can appreciate and understand more fully the following three chapters which overview the systems CASE, DIMUI, and IPSS.

The final three chapters (7 through 9) document the comparative evaluation performed by the autor. Chapter 7 details the criteria chosen upon which the evaluation was based: 1) ease of use, 2) modeler knowledge and understanding, 3) model flexibility, 4) output statistics, 5) program product, 6) portability, 7) hardware characterization, 8) software characterization, 9) data definition facility, 10) data manipulation facility, and 11) device media control facility.

Chapter 8 evaluates each of the three systems by the criteria chosen and justified in Chapter 7. A weighted objective function is derived which shows the DBMS current capabilities of IPSS, CASE, and DIMUI to be approximately 90%, 65%, and 80% respectively.

Recommendations in Chapter 9 consider the appropriateness of each model to DBMS, and how future research should be focused to improve each subject system. Because CASE has no explicit schema or set capability, it is not recommended for use as a general DBMS analysis tool; its use is restricted to timing studies and even there the validation effort is non-trivial. Both IPSS and DIMUI prove to be useful tools for database analysis with much promise for the future. IPSS is a more flexible tool than is DIMUI but DIMUI is far easier to use. These tradeoffs must be carefully considered before a final decision is made concerning future use or support of these systems by AIRMICS.

2. INTRODUCTION TO SIMULATION

To simulate an activity is to mimic its behavior: given the same inputs as the real activity, the simulation model creates outputs that appear indistinguishable from those of the real activity. More explicitly, author R. E. Shannon [1] defines simulation as follows:

> "Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system."

The objective of this chapter is to provide a brief overview of simulation, especially with regard to simulation using computers. Simulation, as a topic in the computer and information science area, is treated as an art rather than a science. As such, it is a tool that must be used with care, else the results will remain dubious to those outside the modeling effort.

2.1 MOTIVATION FOR SIMULATION

Before one constructs a simulation model (which usually represents a large investment in time and monies) sufficient justification must exist to warrant it. Simulation models of all types - from large to small, simple to complex, cheap to expensive - have been constructed in the past, and continue to be constructed in the present. Some of the more important factors which motivate these simulation efforts follow.

1. Cost - a prototype is too costly to build; often we are selecting from a large set of design alternatives, all of which cannot feasibly be built given the cost constraints.
2. Technology - the prototype may be impossible to construct using today's technology but will be available in the future so we wish to model it today.
3. Morality - certain models cannot be executed "real-life" because of moral problems; e.g., pilot crash-landing procedures, simulated war, etc.
4. Time - the actual model executes too quickly or too slowly; a simulator could contract or expand the time interval of simulated time vs. real time. A tree may take 40 years to mature yet a decision about planting/spraying must be made today.
5. Risk - building and bridge stress designs, for example, are much less risky to simulate than to prototype. E.g., placing bumper-to-bumper trucks on a bridge with 80 mph winds to test stress capability carries no risk under simulation.
6. Knowledge - the objective may be to learn more about a system that exists but which we cannot explain. Through simulation we can try to effect the I/O transition of the activity to explain the phenomenon; further we can test it to ensure correctness of our theory.
7. Applicability - often simulation is performed as a last resort: mathematical models are insufficient or undefined; prototypes cannot be constructed without further knowledge about the activity. Construction of a simulation model may lead the way to design/understanding breakthroughs.

2.2 COMPUTER SIMULATION

Simulation is an attractive computer application, as the activity can be executed many times to observe random phenomenon and to thoroughly test the I/O stability of the model. Two basic classes of simulators exist: continuous and discrete. Continuous processes such as heart-beat monitoring, blood pressure, crystal growth, population growth, etc., are most easily modeled using continuous simulation. The input-output cycle is nondiscrete, essentially describing a process that never stops. These normally represent real-time systems in that no clock other than the actual one exists: time contractions or expansions exist only in the sense that the system may not be modeled to the same degree that would reflect an isomorphic image of the critical activity. Analytic models are often applicable to describe these situations as differential equations (linear or non-linear), for example. However, a computer may be required to solve the defined equations in a continuous fashion.

Analog computers are normally used to model continuous processes. The equation(s) to be solved are modeled by sending voltages through the analog network. Each variable identified is represented by a voltage reading between -1 and 1. Output readings can be as significant as the calibration of the analog computer. At any given point in time, the analog computer is wired to represent one equation (or set of equations) and is solving it continuously. The output is normally graphic since it cannot often be utilized in a discretized manner.

Discrete simulators model activities at certain points in time (where time is simulated), but not all points in time. They are characterized by recognizing "events" as they occur against a simulated clock. At any time more than one event is to occur, the simulator does each task associated with each event sequentially, but without advancing the simulation clock. In this way parallel or concurrent processes can be modeled sequentially.

Digital computers are used for discrete simulation modeling at either the macroscopic or microscopic level. Given this powerful tool, large complex simulations can be effected that would be impossible to perform otherwise. However, the digital computer is a sequantial machine, not naturally inclined to perform simulation concepts such as queueing, interrupts, parallel processing, etc. Software facilities (simulation tools) must be made available to the user so that he can create and execute a simulation model.

2.3 SIMULATION TOOLS

There are many software facilities available to the modeler to ease the simulation task. His choice from among this selection will depend upon his own background in modeling and programming, and the applicability of the software facility to the problem at hand. As Figure 2-1 illustrates, one may choose a general programming language such as FORTRAN or PL/1 and construct a model from scratch. On the other hand, an existing specialized programming language with built-in simulation capabilities such as GPSS or SIMSCRIPT may be utilized. Lastly, if the simulation is focused upon simulating computers, then a computer simulation system such as CASE, DIMUI, or IPSS may be desirable.

While this report focuses upon the latter systems, a short explanation of modeling in the other two environments will serve to place simulation of computer systems in the proper perspective.

2.3.1 *GENERAL PROGRAMMING LANGUAGES*

Languages such as FORTRAN and PL/1 were developed to enable the user to perform general tasks using the digital computer. These languages provide the user a program and data area. Data input and output are performed via the data area and calculations such as multiply, square root, divide can be directed in any sequence of instructions defined by the user. The advantage of the digital computer over the analog computer is that it is programmable; outside intervention is not required to change the stream of isntructions to be executed by the computer. The basic FORTRAN capabilities available to the user are vectors of up to 5 dimensions and subprograms. Vectors enable one to reference element VALUE(I) in a statement and have it refer to, during execution, the I-th item of vector VALUE where any other statement can alter variable I. Subprograms enable modular program definition and isolation of program code. These are the main two features of FORTRAN that enable sophisticated programs to be defined and executed.

Thus we see that the general programming languages are not particularly simulation-oriented. The user must use vectors to simulate queues. For instance a departure (front person exiting the simulated queue) is modeled by moving the 2nd through the last element of the vector up one position. We see that one is using static elements (vector sizes in FORTRAN are fixed) to model dynamic entities - queues. FORTRAN supplies no clock to the user - he must himself simulate it. If the clock reads 4:00 and the next event does not occur (future events queue) until 4:20, then we simply add 20 minutes to the clock and proceed. While this all can be done using the facilities of a general programming language, it is a non-trivial task that requires one to model not only the process of interest (the simulated activity) but also to work within the confines of the language facilities available. Furthermore, the completed model represents a particular system and normally

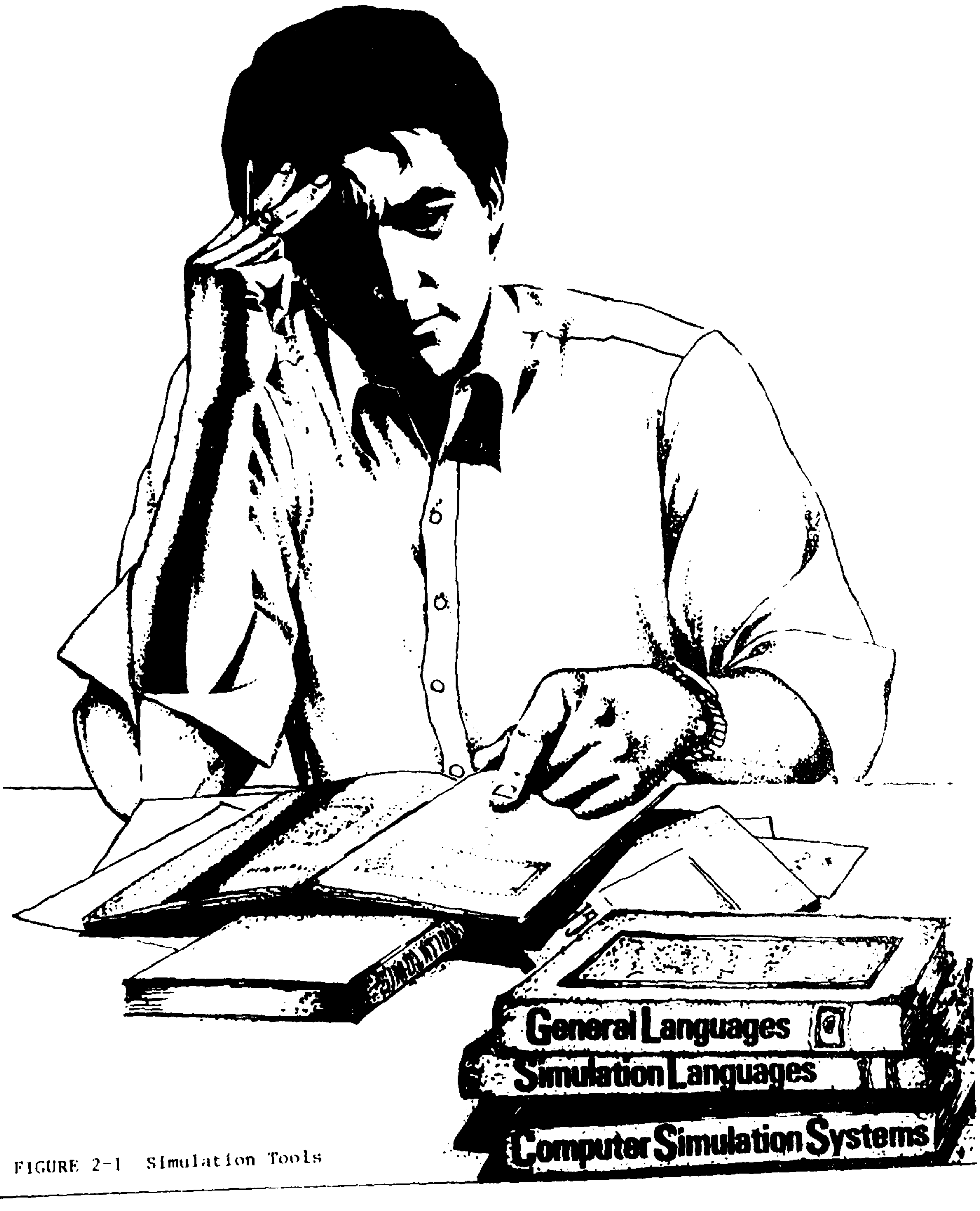


FIGURE 2-1 Simulation Tools

is not generalizable or flexible. Significant modifications are required to model related systems.

2.3.2 GENERAL SIMULATION LANGUAGES

About 10 years after the development of general programming languages it was recognized that other larguages, with particular focus upon simulation, would be valuable to the computing community. Several languages were developed, to include GPSS and SIMSCRIPT. While these languages are somewhat dissimilar, the fact is that both are general-purpose simulation languages. They provide a comfortable environment so that the user can do large simulations with relative ease. To contrast these languages to the general purpose languages FORTRAN and PL/1, the below comments consider queues and clock time as noted in the previous section.

The General Purpose Simulation System (GPSS) language was developed by IBM in the mid sixties [2]. It provides a complete programming language (unfortunately without the subprogram concept) that includes the dynamic entities of QUEUE and FACILITY, an implicit clock, and TABULATE commands to gather and report statistics on idle time, etc. Hence a waiting line is characterized by the QUEUE and DEPART commands (with PRIORITY if needed); a FACILITY can model, for instance, a ticket-taker or some service, with ENTER and LEAVE commands; the clock is updated using the ADVANCE command; the GENERATE command automatically creates arrivals in accordance to a user-defined distribution of inter-arrival times.

Hence the process of model building and execution is considerably eased with the help of simulation langauges such as GPSS. However, when it comes to simulating complex computer systems the task is still incredibly difficult. There are no vehicles available to easily characterize data structures, physical devices, access paths (no vectors in GPSS, per se), etc.

While large models of computer systems have been developed using GPSS, they are difficult to validate and do not provide sufficient statistics. GPSS is not a host language within which the user can easily imbed additional commands in FORTRAN or any other language. Hence, other computer-related simulation facilities have been developed to meet this particular need.

2.3.3 COMPUTER SIMULATION SYSTEMS

A number of computer simulation systems have been developed recently by industry and researchers alike. The objective of these systems is to provide a vehicle for the evaluation of various hardware and software systems and alternatives. Many of these systems are not available to the general public, while others, like CASE, SCERT, and ECSS are purchase items. Other systems, such as DIMUI and IPSS, are research tools developed at universities that are not, as of yet, completed systems available for immediate public use. This study is restricted to considering three computer simulation systems with database modeling capabilities: CASE, DIMUI, and IPSS. Each of these systems offers extensive aids to the user to ease the modeling task.

CASE (Computer Aided Systems Evaluation) uses primarily a "black box" approach to simulating computer systems. The user definition of the hardware model and files is done completely through user-defined parameters. While this certianly eases the modeler's work it also isolates him from the actual simulation model and makes it extremely difficult, if not impossible, for the user to modify the CASE model.

The other user-defined input to the CASE simulator is a definition of the user workload with adequate CASE-provided language constructs to describe user application programs. A typical CASE simulation entails many executions of the Independent Processing Analyzer (IPA) followed by one execution of the of the Concurrent Processing Analyzer (CPA). Parameters are available for user-choice of statistical output, but the output is restricted to the pre-defined capabilities of CASE. The modeler is not capable of modifying the CASE internal model, to include statistical capabilities. CASE is, nonetheless, easy to use, cheap to operate, and quick to set-up and run.

DIMUI (Database Implementation Model Using Induction) also uses a "black box" approach to simulating computer systems. The premise of DIMUI is that the computer system is basically I/O bound (i.e., input/output operations are the constraining factor in processing the workload). In light of this, DIMUI is designed to focus upon data flow. A primary objective of DIMUI is to provide a database modeling facility so that alternative database designs can be evaluated through simulation.

The user provides two major inputs to the simulator, using DIMUI-provided language constructs. First is a definition of the logical database, in the form of schema definitions. Presently, this is input in the form of a Data Definition Language (DDL), which was just recently developed for DIMUI. The second input is comprised of DIMUI DML (Data Manipulation Language) statements used to characterize the user application programs. This is a more powerful construct than that provided by CASE with regard to database analysis, but is still inadquate with regard to the output statistics provided by DIMUI. As is also true with CASE, the user cannot easily modify the DIMUI-defined computer system components.

IPSS (Information Processing System Simualtor) provides a methodology and a language for user characterization of the entire system to be simulated. Hence its approach is much more extensive (and difficult) than that of CASE or DIMUI. It is more like a specialized simulation language (whereas GPSS and SIMSCRIPT are general) for defining and simulating computer information processing systems. The IPSS methodology provides a structured, top-down, modular approach to model definition. Using this viewpoint, the user understanding of system components leads naturally to definition of the IPSS model components.

The IPSS language constructs are particular to computer systems (hardware and software) and database systems. IPSS provides DDL and DML statements, hardware characterization statements, queueing and timing capabilities somewhat like the GPSS constructs, and statistics-gathering statements. Using the IPSS language, and imbedding FORTRAN statements whenever desired, the modeler defines the IPSS model components: System Resources, Storage Structure, Data Base Access, Data Structure, and Request Stream. Thus the IPSS is a design tool, providing a language and a methodology for constructing simulation models of information processing systems.

3. MODELING DATABASE ACTIVITIES

The primary objective of this research project is to evaluate the database modeling capabilities of the CASE, DIMUI, and IPSS simulation facilities. This is of great importance because the trend in computing is definitely towards database systems in all applications involving large databases. This chapter provides a brief overview of the three major theories of database so that those modeling aspects necessary to simulate database activities can be appreciated and understood in the proper perspective.

3.1 MOTIVATION FOR DATABASE MANAGEMENT SYSTEMS

As computers have evolved and become more complicated and sophisticated, so the information flowing into and out of the system has increased, both in sheer magnitude and in complexity. The large costs entailed in software development are lost during evolution unless it can be done in a generalized manner. Files created by one application must be available to other processes else redundant data results, which increases the already huge data storage requirements of a computer system. Database management systems were developed to respond to the need to more efficiently handle the growing data flow in information processing systems.

A database management system (DBMS) offers eight assets to the data processing enviornment that are of such value that they provide the motivation for the development of such a capability.

1) Integration

The DBMS considers the entire collection of files and data as a single, integrated database. The effect of this is to enable a user application to refer to files created by other applications without great difficulty; all data belonging to the computer system is accessible by the user. Data redundancy is greatly reduced since it need exist only once to be accessible to all users.

2) Physical Independence

The DBMS enables the user to refer to data logically, not physically. Hence the user is insulated from format changes, etc., in the actual database; he need not know how the data is stored, only that it exists. Thus, much nitty-gritty knowledge which is in the hands of only the data creater need not be forwarded to the data user. This removes both JCL and format difficulties from the users domain.

3) Logical Independence

The DBMS provides, through the concept of schemas and subschemas, the capability for diverse users to view the database in a manner local to their needs. Hence the DBMS translates the global view of the database to the local view relevant to the application at hand. Logical and physical independence cut the costs of creating, modifying, and updating application programs appreciably.

4) Security

By providing schema (templates) for viewing the database, the DBMS can restrict user classes to certain views of the data. Since DBMS implies an integrated system, security is a mandatory facet of the process. Although the data exists, a user may be restricted, for example, to accessing only that data less than or equal to his security classification. The DBMS, in acting as a translation buffer between user data requests and actual data accesses, can screen all requests before data access is accomplished.

5) Privacy

The explosion of data stored in computers has raised the problem of invasion of privacy. If I bounced a check should any arbitrary browser be able to learn this? (If it is stored somewhere, then many people calim they have a right to be informed of the contents.) If a programmer is running an aggregate survey job on salaries (e.g., how many employees earn over $15,000, etc.) should he be able to learn how much a particular John Doe makes? The DBMS in a manner much like the security measures, can screen the data so the programmer can see the SALARY field, but not the NAME field of the record. The advantages of this are that data integration can be accomplished without compromising the creators of the data.

6) Integrity

The DBMS ensures better data integrity (correctness) in two ways. First, data redundancy is virtually eliminated, thus reducing to a fraction the problem of duplicate records that, in fact, are different in some data item respect. Update costs are reduced since most data items are stored only once in thedatabase. Chances of data modification by unauthorized users are lessened due to DBMS privacy and security facets. The DBMS can further ensure integrity by knowing what ranges are permissible for given logical data items. If SALARY ranges from 10 to 100K, then any value outside this range is suspect and can be made known to the system by the DBMS.

7) Standardization

The DBMS creates one (standard) logical definition of the database. Although the user can form his own local logical view of the data, he cannot alter the physical data format. The standards are in the hands of the Database Administrator and he is in complete control.

8) Modularity

The DBMS acts as a buffer between user data requests and database accesses. The DBMS activity, as such, can be performed on the same computer the user is running his application, or on a microcomputer imbedded within the computer system, or on a different, specialized backend machine. There are great CPU advantages to running the DBMS on another CPU: throughput can be enhanced significantly. The IOCS (Input/Output Control System) is now under control of the DBMS rather than the operating system of the CPU running the application program.

So we see that there are many advantages that accrue under DBMS that make its consideration in future (if not present) systems almost obligatory. Given that we are considering a DBMS, then we must determine what type of database to construct. There are three major types of database models and we address these in the following sections.

3.2 THE RELATIONAL MODEL

Relational concepts and early models were considered as far back as 15 years ago. Recently, research by E. F. Codd [3] created new interest in the relational model. The value of the relational model lies in its inherent simplicity, and the resultant logical ease with which user queries can be processed. The relational structures to be discussed are logical; they need not be physically stored as such. Relational structures are valuable for the convenience they provide to application programmers and casual users of the system. As a relational database grows and is modified, its complexity remains the same.

Relational data is stored in a normalized, tabular form. Relational tables are logically constructed that consist of n-tuples. Each row of a table represents a relation of degree n. Since no hierarchy exists (as does in both the network and hierarchical models to be next explained) the relational database is considered a "flat" file. The result is a relational database consisting of sets of 2-dimensional tables that represent n-ary relations, and primitive operations for extracting and joining columns and constructing new tables representing all data items satisfying some desired relationship.

Let us consider a simple relational example; one that we shall duplicate in the consideration of both hierarchical and network models. The database is comprised of SUPPLIER records and PARTS records. Associated with each supplier are all the parts produced by this supplier; a part may be produced by more than one supplier. Figure 3-1 illustrates the SUPPLIER and PARTS record types.

A relational database describing the supplier parts data and interrelations would consist of three relations, each of which is a logical table as instantiated in Figure 3-2. The Supplier table entries relate each supplier name to its id, location, and manager. Each Part table entry relates a part to its id, color, size, and cost. Thus the Supplier and Part tables hold the data contents of the database. The additional relationship between suppliers and parts is a generated relationship which binds parts to suppliers and vice versa. This relation is described by the Sup-Prt relational table. This table consists solely of record keys: additional supplier or part data must be obtained via the Supplier or Part relation.

Supplier:

id$_s$	name$_s$	location	manager

Part:

id$_p$	name$_p$	color	size	cost

Sup-Prt:

id$_s$	id$_p$

FIGURE 3-1. Supplier and Part Record Types and Relations

Supplier Relation

id$_s$	name$_s$	location	manager
A	Ford	D.C.	L. Roth
B	GM	Detroit	B. Jones
C	Datsun	Japan	H. Zuki

id$_p$	name$_p$	color	size	cost
1	bolt	brown	3/8	.29
2	bolt	gray	3/8	.54
3	tire	black	E14	51.
4	tube	black	E14	3.49
5	door	blue	30	128.

Sup-Prt Relation

id$_s$	id$_p$
A	1
A	3
A	4
B	1
B	2
B	5
C	3
C	4

FIGURE 3-2. Example Relational Instance

One example use of the relational database model will be illustrated, and used for comparison and contrast with the equivalent network and hierarchical models to be described in the following sections. Let us use the relational database to find the answer to the question: "Which suppliers produce gray bolts?".

1) Using the Part relation we find that gray bolts do exist, and are identified by $id_p = 2$.

2) Using the Sup-Prt relation, we sequentially scan the part id column, and for each $id_p = 2$, we expand our new relational table. As shown below, only supplier B produces gray bolts, so the answer (a new relation) has only one row entry.

Supplier of Gray Bolts Relation

B	GM	Detroit	B. Jones

Note that the answer in this case (representing a new relation) represents a logical subset of the Supplier relation, and was derived using the existing Supplier, Sup-Prt and Part relations. Typical work with a relational database consists of extracting or adding rows/columns to/from existing relational tables to achieve the desired result: a table all of whose entries satisfy the requested relationship.

Thus, using "flat" files/tables, the relational model can describe, in this case, a many to many hierarchical relationship without structuring the data hierarchially. The representation of relations as physical 2-dimensional (flat) tables is logical: no assumption about actual data storage is implied or required. The user visualizes the flat relations and works accordingly. The database management system must effect the data storage transformations required by user requests to extract/store data and to and/or/not the logical tables.

The beauty of the relational system is that, given this logical view, a primitive logical calculus can be employed to enable users to easily describe what actions should be done (at the logical level) to process a relational query. A disadvantage of the relational model is that query processing can create a number of huge relational tables representing partial answers; these directly increase the dynamic computer storage requirements of the DBMS. Optimization of relational query processing is a difficult task not yet completely resolved.

3.3 THE HIERARCHICAL MODEL

The second database model of importance is the hierarchical model [4]. As the name implies, all data is structured hierarchically, in a tree-like form. This design is in direct contrast to the "flat file" principle of the relational model. Commercial systems, such as System 2000, have been implemented based upon the hierarchical model and have been mareketed successfully.

Hierarchical repersentation of data proves very useful, as most databases are so naturally ordered. For example, an insurance company database of policies is naturally structured in the hierarchy: the corporation has regional offices; each office has its assigned insurance salesman; each salesman maintains the policies of his clients. The hierarchy shown in Figure 3.3 is tree-structured: the relationships within it (e.g., salesman in a given region) are strictly 1:m (one to many). Each record-type at a given level has but one owner. This provides a natural partitioning of the database of which we will take great advantage. When doing a search of the database, going from one level to the next isolates the search to a specific subset of the database. For instance, if the query specifies client data in only region z, then all

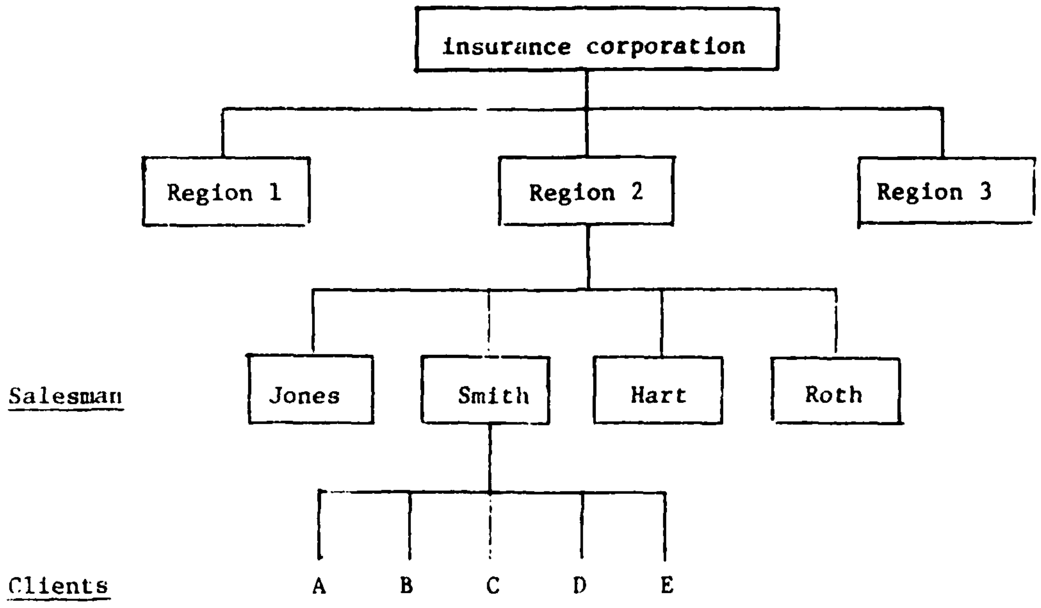


FIGURE 3-3. Hierarchical Representation

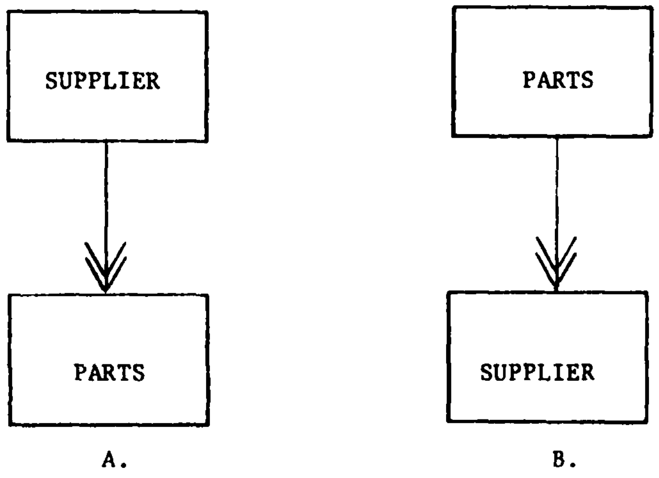


FIGURE 3-4. Hierarchical Supplier-Parts Model

client data attached to other areas will be avoided, saving significant CPU time in the search.

Let us now consider the Supplier-Parts example of section 3.2 so that the hierarchical model can be conveniently contrasted to the relational model. Figure 3-4 illustrates the two logical models that could be employed: either Parts are subservient to Supplier or vice versa. As there are many more distinct parts than distinct suppliers, it is clear that version A of Figure 3-4 is the desired schema. This means that the partitioning resulting from our structure will be with respect to Supplier, and not to Part-types (which may repeat).

We illustrate in Figure 3-5 an instance (equivalent to the relational instance of Figure 3-2) of the hierarchical model of the parts-supplier database under discussion. Note that the relation "supplied by" is stored implicitly in the structure of the database. This is equivalent ot the Sup-Prt relation necessary in the flat relational model example. Note also the records Brown Bolt, Black Tire, and Black Tube are repetitive in the instance. This is necessary due to the natural partitioning requisites of a hierarchy: no record may have two "owners".

The sample schema definitions of Figure 3-6 illustrate the manner by which the user defines the Supplier and Part records, and the Part set, each of which is comprised of Part records associated with each Supplier record. The hierarchical structure need not be confined to two levels: the Part records may "own", for instance, the Stock records which show where and in what quantity this part is available.

Thus in the hierarchical model, a record may be the owner of one set and the member of another; but by definition the owner is unique and at the next level above; the members are at the next level below. We also noted the existence of multiple copies of records. This is logical duplication; its physical manifestation is implementation-dependent.

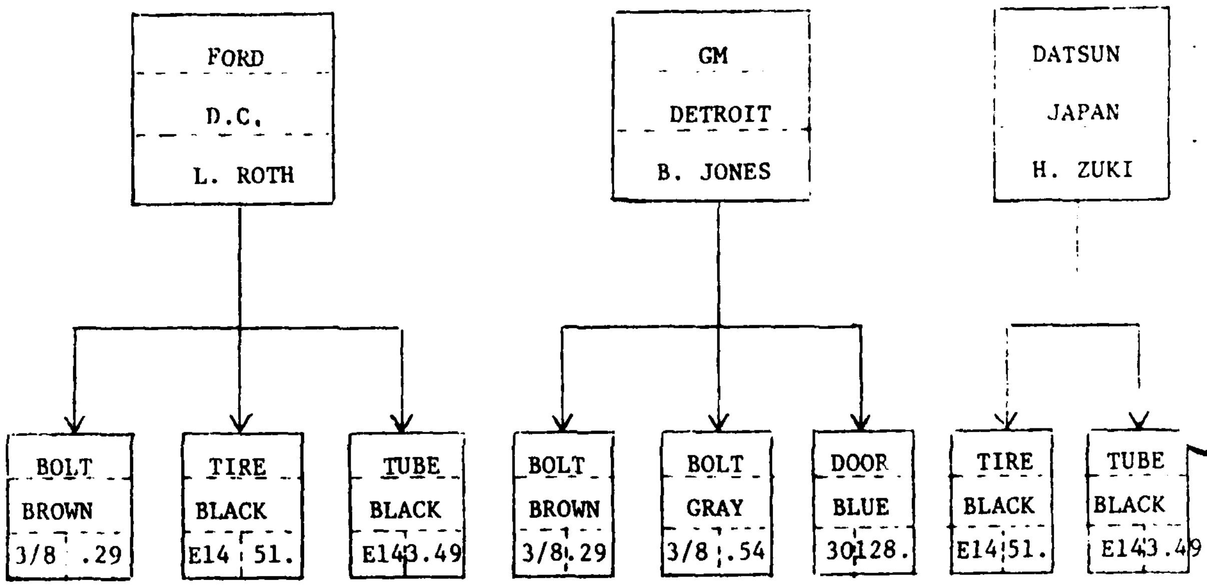


FIGURE 3-5. Example Hierarchical Instance

Schema Definitions

- Supplier "contains" name, location, manager

- Part "contains" name, color, size, cost

Set Definitions

- PARTSET: Each Supplier "owns" Part*

*Defines 1:M relationship; PARTS are "members" of PARTS set associated with each SUPPLIER.

FIGURE 3-6. Schemas and Sets Example

To complete our discussion of hierarchical models, let us consider the sample question we posed in the relational example: "Which suppliers produce gray bolts?". In the hierarchical model shown in Figure 3.5, the search proceeds as follows:

```
1) for next SUPPLIER record do the folloing:
2) Find first member of PART set;
   If PART.name = Bolt and PART.color = gray
   then list SUPPLIER and go to 1); else
3) Find next member of PART set; if none go to 1);
   If PART.name = Bolt and PART.color = gray
   then list SUPPLIER and go to 1); else repeat 3).
```

This algorithm describing the search procedure is very similar to the application programmer code used to do the search; the power of DBMS becomes more apparent. The user logically follows through the hierarchy using set and member types which the DBMS translates into the actual accesses required.

This example illustrates that the hierarchical structure chosen is sub-optimal for this particular query. It can be accommodated, but requires a search of nearly the entire database. In this example, all but one element of the parts data (gray door) was examined. However, a query considering GM-supplied bolts would immediately preclude FORD and DATSUN parts examination. This problem may be avoided in the network model, as the unique set-member relationship is not required.

3.4 THE NETWORK MODEL

The network model was endorsed by the CODASYL Data Base Task Group [5] as a standard database model with standard data definition (DDL) and data manipulation (DML) languages. It is an extension of the hierarchical model which provides greater complexity and flexibility in possible data relationships.

While the relational model employs flat files, and the hierarchical model restricts owner-member relationships to the 1:m category, the network model provides the direct capability to characterize and manipulate data bases comprised of many to many (m:m) owner-member relationships. Figure 3.7 illustrates this basic conceptual difference in data relationships. The relaxation of the 1:m owner-member restriction implies that a member may have more than one owner. Hence we evolve from a strict partitioned tree to a semi-partitioned network capability; the application programmer can go across partitions using other than hierarchical relations.

The network model thus provides more flexibility to the user by enabling richer relations to be established. This provides the capability to access the database in a much more powerful and flexible manner and removes the need for data redundancy.

The Parts-Supplier example we have used in the previous two examples is illustrated for the network model in Figure 3.8. We see that, in addition to the Part set equivalently represented by the hierarchical model, we can also go backwards, in effect, and represent a Produced.By set to show which suppliers produce a given part. We thus have access to the data by either Part or Supplier name (as we did in the relational but not the hierarchical model) and hierarchies in both directions via the two sets (as did neither the hierarchical nor the relational model).

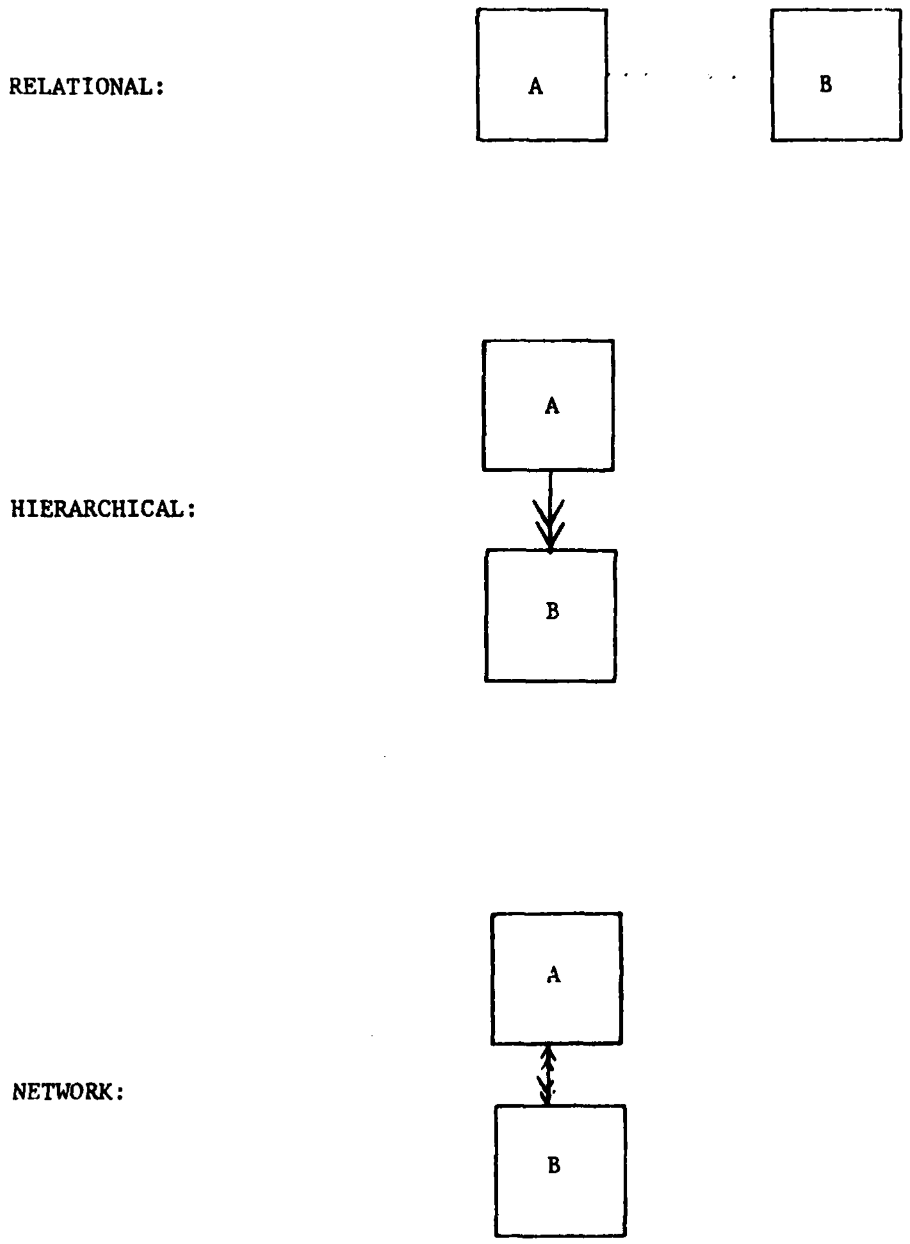
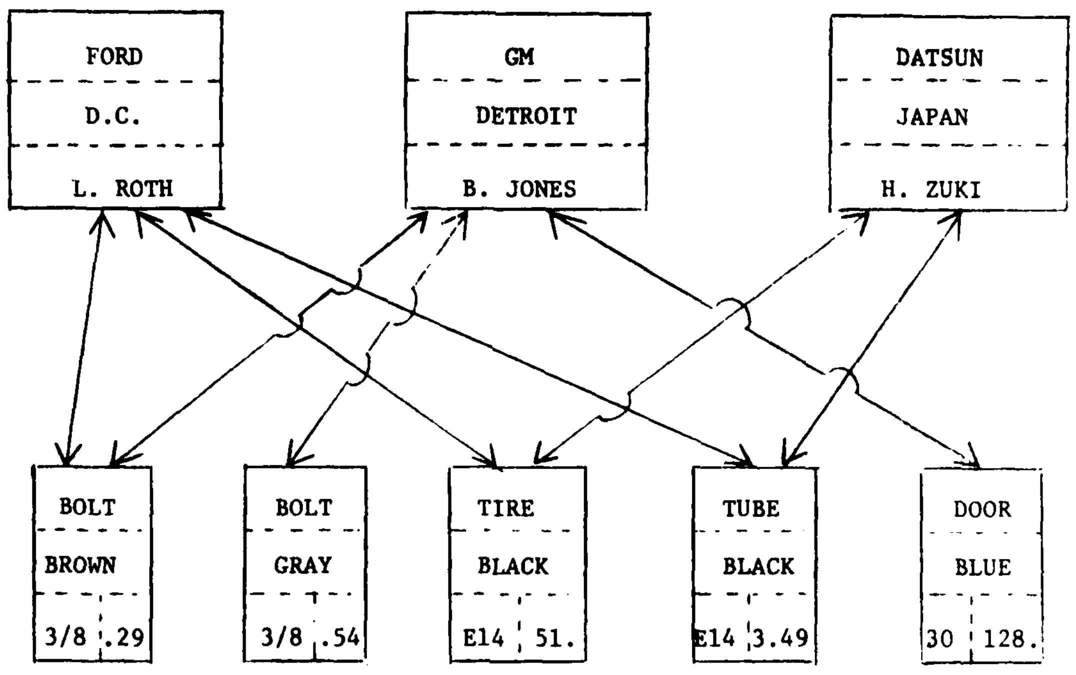


FIGURE 3-7. Data Model Representation



<u>Two Sets</u>: m-m relationship

- PARTS shows what is produced by each supplier
- PRODUCED_BY shows what suppliers produce a given part.

FIGURE 3-8. Example Network Instance

In a sense, we can have our cake and eat it too, although clearly this is a more complex structure and we can expect the DBMS system to be correspondingly more expensive and complex. Data redundancy is reduced as we store neither the relations (relational model) nor duplicate set members (hierarchical model).

We can now proceed to analyze the processing of our example query, "Which suppliers produce gray bolts?", in the network model. Using the Part set we find one gray bolt and then use its Produced.By set to enumerate all other suppliers. Here is our network algorithm:

```
1)  for next SUPPLIER record do the following:
2)  find first member of PART set;
    If PART.name = Bolt and PART.color = gray
    then go to (4) else;
3)  find next member of PART set; if none go to 1);
    If PART.name = Bolt and PART.color = gray
    then go to 4) else repeat 3);
4)  List all members of PRODUCED.BY set for this PART.
```

While this algorithm is longer in definition than was our algorithm for the relational and hierarchical models, search time is minimized. For once we find the part, the Produced.By set points to all producers of that part. In this example, only 2 of the 5 parts are examined, and only one Part set is examined. Thus the network model can lead to not only more flexible processing of data, but also more efficient processing.

3.5 SUMMARY

This chapter has provided an overview of the three foremost database methodologies for database structure and characterization of data relations. Each has its advantages and disadvantages. It was not the intent of the author to espouse preference of one over the other, for all three models are in use and probably will continue for some time.

For this reason, it is necessary that any database simulator provide a mechanism to model all three models if it is to be considered completely general. As hierarchical models are restricted network models, the great deviation, with regard to simulation modeling, comes with relational modeling.

The following three chapters consider the three computer database model simulators which are under consideration. These system synopses will also further educate the reader in database concepts; hence specific database modeling evaluation by the author is postponed until after these three systems have been discussed in detail.

4. CASE OVERVIEW

The Computer Aided System Evaluation (CASE) package is a simulation design facility which can be used to model computer systems in both batch and concurrent processing modes [6]. CASE is a proprietary software package developed, maintained, and sold by TESDATA Systems Corporation. One of its clients is the U. S. Army, which uses it almost exclusively to simulate their batch processing enviornment. CASE is a completed, documented product that provides a wide variety of statistics to support the simulation of computer systems. It has been implemented successfully on IBM, Honeywell, Univac, and CDC hardware systems.

CASE was designed to minimize the users work requirements; it is easy to use. The CASE software package simulates the computer system and information processing as defined by user parameters. This "black box" approach to simulation has both advantages and disadvantages. On the pro side, the user need not be a simulation or computer systems expert. User definition of inputs to CASE is straight-forward and quick. The user/modeler writes no software to characterize the system; thus it is cost-effective if it can produce the outputs desired. On the con side, the modeler cannot modify the CASE simulator. The flexibility of CASE is restricted directly to the parameters available to the modeler to define the simulation. Software code, FORTRAN or otherwise, cannot be imbedded by the user into the CASE simulator. As the modeler does not define or modify the CASE software simulater, he runs the risk of running a simulation he does not understand. Technical documentation of CASE internals is virutally non-existant; the CASE source code is poorly documented and not available to the general user.

Simulation using CASE, from the modeler's viewpoint, does not provide additional understanding of the system being simulated at the internal level; knowledge generated is at the input/output level with the CASE "black box" in-between. For those simulations appropriate to CASE, the package is highly valuable for providing quick answers with a minimal amount of modeler work required.

4.1 THE CASE METHODOLOGY

The CASE methodology is to provide a vehicle (not a language) for computer systems simulation such that different computer systems and processes can be modeled by user definition of the available parameters to the CASE simulator. Output statistics are also parameterized and automatically produced if requested by the user.

The internal CASE methodology (i.e., CASE's view of the system-processes interaction) is completely hidden from the user, and documented at a non-technical level. The user is instructed how to use CASE, but not told how it works. Hence the modeler is using a simulation model developed by others that he may not understand. This results in two shortcomings: 1) increased CASE use does not lead to increased knowledge of the system being simulated; and 2) user analysis of I/O reports may be wrong due to lack of understanding of the CASE black box simulator.

The CASE methodology for modeling is to characterize the particular computer system and processes under consideration by user input parameters, and to simulate the system so defined. Figure 4-1 illustrates a high level view of the CASE system.

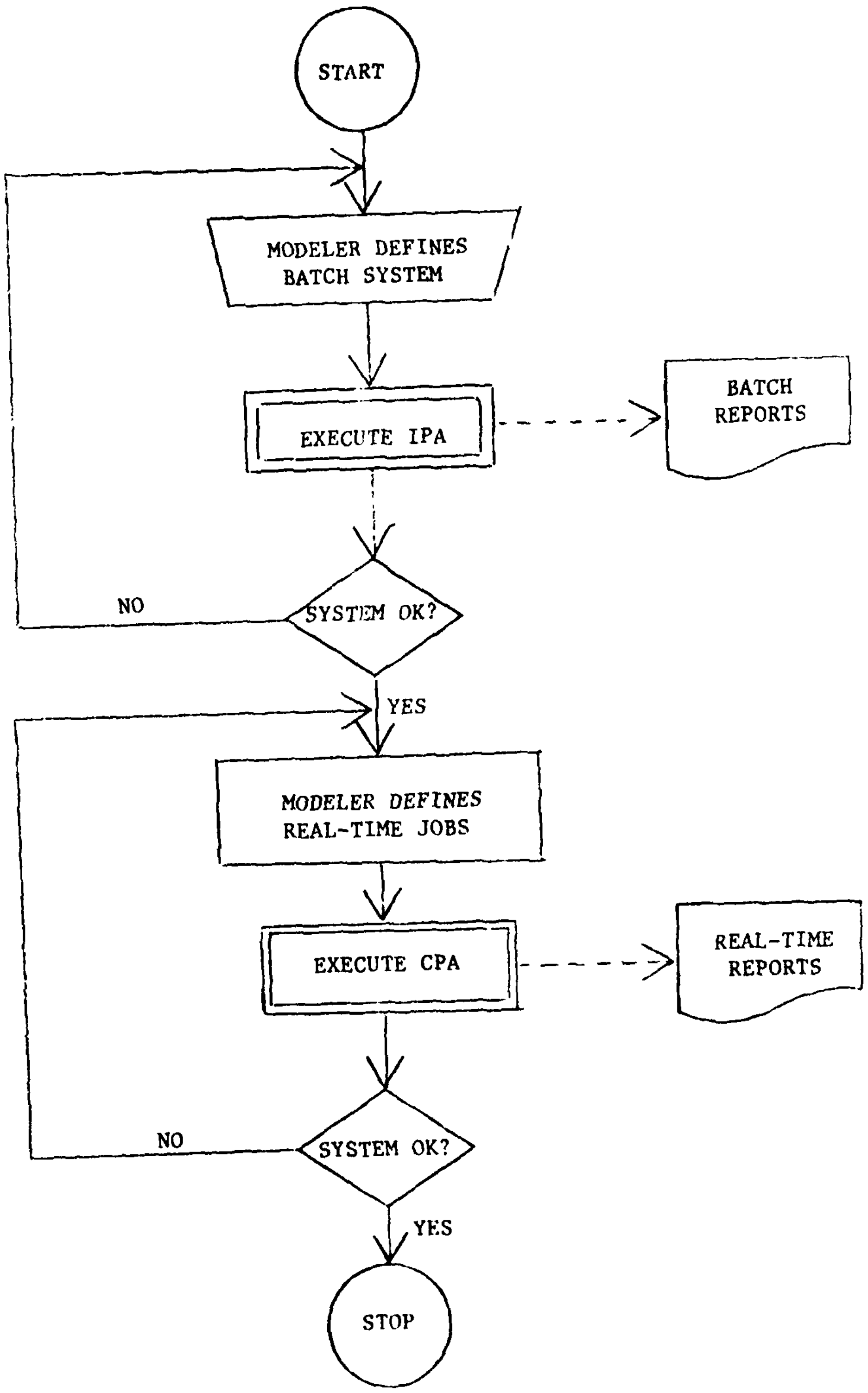


FIGURE 4-1. Case System Overview

Modeler input to CASE consists of four classes of user parameters;

1. Simulation Controls - general control parameters, output statistics requested, frequency of reporting, etc.;
2. Equipment Definition - number and types of hardware devices, their interconnections, executive software (sorts, compilers, etc.);
3. File Definition - volume, media, organization, record and block-size, master file designation, etc.; and
4. Process Definition - application programs, file references, number and type of instructions, batch and real-time.

These parameters serve to define the batch system and the real-time jobs if applicable. Equipment and process definition is carried out with the support of the CASE library, which is an extensive user aid.

The CASE simulator is divided into two distinct modules, denoted IPA and CPA. The Independent Process Analyzer (IPA) is a FORTRAN batch simulator (220K) which simulates the execution of user-defined jobs in a stand-alone environment. Space requirements, internal processing times, I/O file times, and overall execution times are computer independently for each user-defined job on the user parameterized computer system simulated by IPA.

The Concurrent Process Analyzer (CPA) is a FORTRAN real-time simulator (130K) which simulates the execution of the job stream output from IPA in a multiprogramming environment given the modeler-defined real-time process execution and scheduling constraints. CPA provides activity statistics by time interval and reports the derived run schedule and run execution. Utilization statistics are also provided to the modeler for optimization analysis.

Both IPA and CPA are designed to be executed iteratively until the modeler obtains the results he requires. IPA can be run and the results used without further execution of CPA whatsoever. This is the appropriate mode of operation for simulating batch programming systems. Once the IPA simulation phase is complete, CPA may be invoked iteratively to produce real-time multiprogramming enviornment statistics. CPA cannot be run without IPA having been run previously to create the run-time requirements of the user jobs to be simulated.

Further discussion of the CASE simulation facility will be broken down into four natural categories: Irput, IPA, CPA, and Output. Since discussion of model inputs requires some knowledge of the CASE simulator, we defer input considerations until IPA and CPA have been further discussed. Lastly, section 4.5 will overview the types of output statistics provided by CASE.

4.2 THE INDEPENDENT PROCESS ANALYZER (IPA)

The modeler must have some understanding of the internal IPA logic in order to be able to fully appreciate and understand the output statistics resulting from IPA execution. Full delineation of IPA capabilities and processes is not contained in the CASE User Guide. In fact, this user guide is focused completely upon user-definition of CASE inputs.

This author, with great effort, extracted from the CASE manual and USACSC personnel familiar with CASE a reasonably detailed explanation of IPA. The logic process of IPA is illustrated in Figure 4-2, and is discussed herein. The explanation is separated into 11 steps as noted in the flowchart of Figure 4-2.

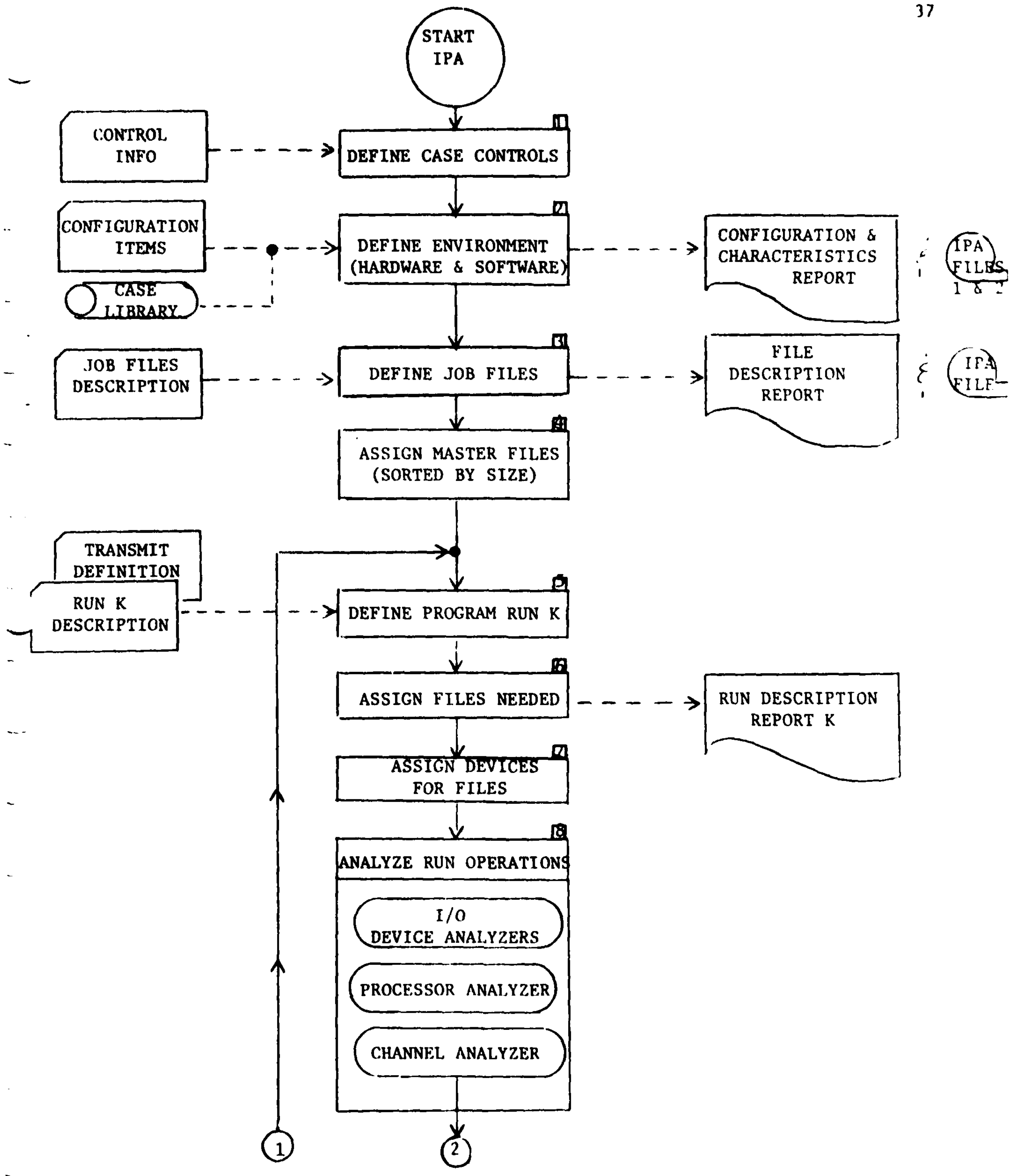


FIGURE 4-2. CASE/IPA Processing: 1 of 2

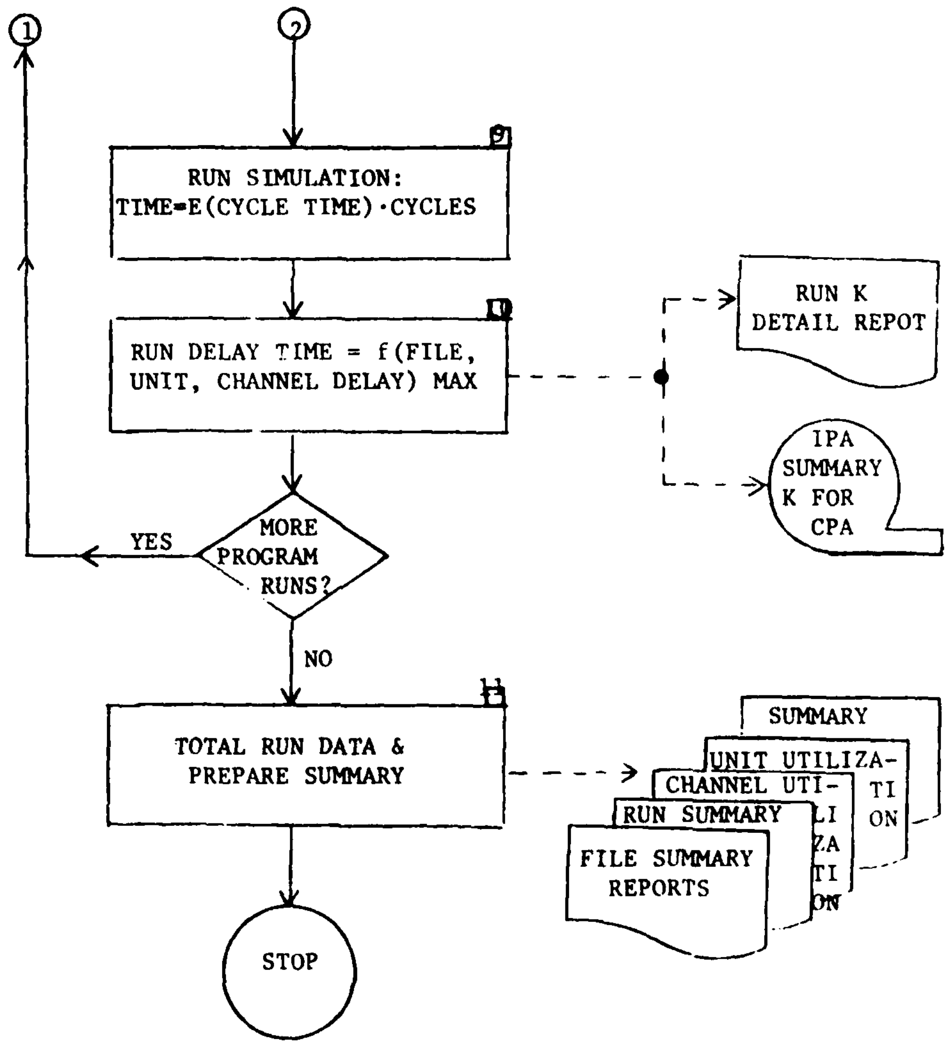


FIGURE 4-2 (Continued). CASE/IPA Processing: 2 of 2

1. CASE controls are defined which describe the reports to be generated; the I/O units for CASE general installation data; processing parameters to include file assignment, sort controls, run generation, and file reassignment.
2. The environment (hardware and executive software) is defined for items such as processor, memory, software, I/O units, controller, and channel by type, manufacturer, model, quantity, reference number, and interconnections. The CASE hardware/software library is used to complete these definitions. Reports emanating from this step are of a compressed form and relevant only to the modeler to verify his enviornmental definition; they are thus internal reports only.
3. The job files (all files in the workload) are defined by user input and converted into tabular form for internal CASE usage. If user-requested, the file Description Report is generated to echo the user input.
4. Master (permanent) files are now assigned to the I/O devices. This is accomplished by first sorting the files by size and then using user designations (immediate access, tape, etc.) to determine appropriate file assignments given the available space on each storage device.
5. Run processing (steps 5 through 10 in loop 1) begins with program run definition given the user input describing this run. CASE runs are classified as general, translation, or sort runs, consisting of program and files. These definitions are detailed in section 4.4.

6. The file requirements of this run are determined, to include intermediate files for sorts, for example.

7. The files required to support this particular run are assigned to the devices with the CASE objectives of minimizing the number of packs required, balancing space allocation on immediate access units, and minimizing channel conflicts.

8. The operations comprising the run are now analyzed by CASE to determine total component time, average basic task time, total channel time, average channel time for basic task, space requirements, and blocking. The operations are analyzed independent of one another by one of the these CASE analyzers: I/O device, Processor (CPU), or Channel.

9. Run simulation is performed to determine the batch execution time of this run. This is done by cycling through the basic tasks, using probabilities of execution, and using the appropriate output of a CASE analysis of the basic tasks. The cycle is re-run until its average execution time stabilizes; from this the run execution time is determined.

10. To complete the analysis, run delay time is computed for each component and the maximum of these is distributed over the files to mirror actual predicted run execution time. If requested, the run detail report is generated at this time.

11. After all runs have been simulated independently, totals are computed and output as the reports shown in Figure 4-2.

The modeler now analyzes all CASE IPA outputs to determine the suitability of his simulation definition to his needs. If errors or alterations of input data require changes, then the IPA module is executed iteratively until successful conclusion. At this time, the modeler has a documented simulation of the execution of his defined program runs against his defined files in the defined computer environment. This may be sufficient for his needs. If, however, he wishes to model the runs in a concurrent environment (as opposed tc batch) then the CPA module must now be executed.

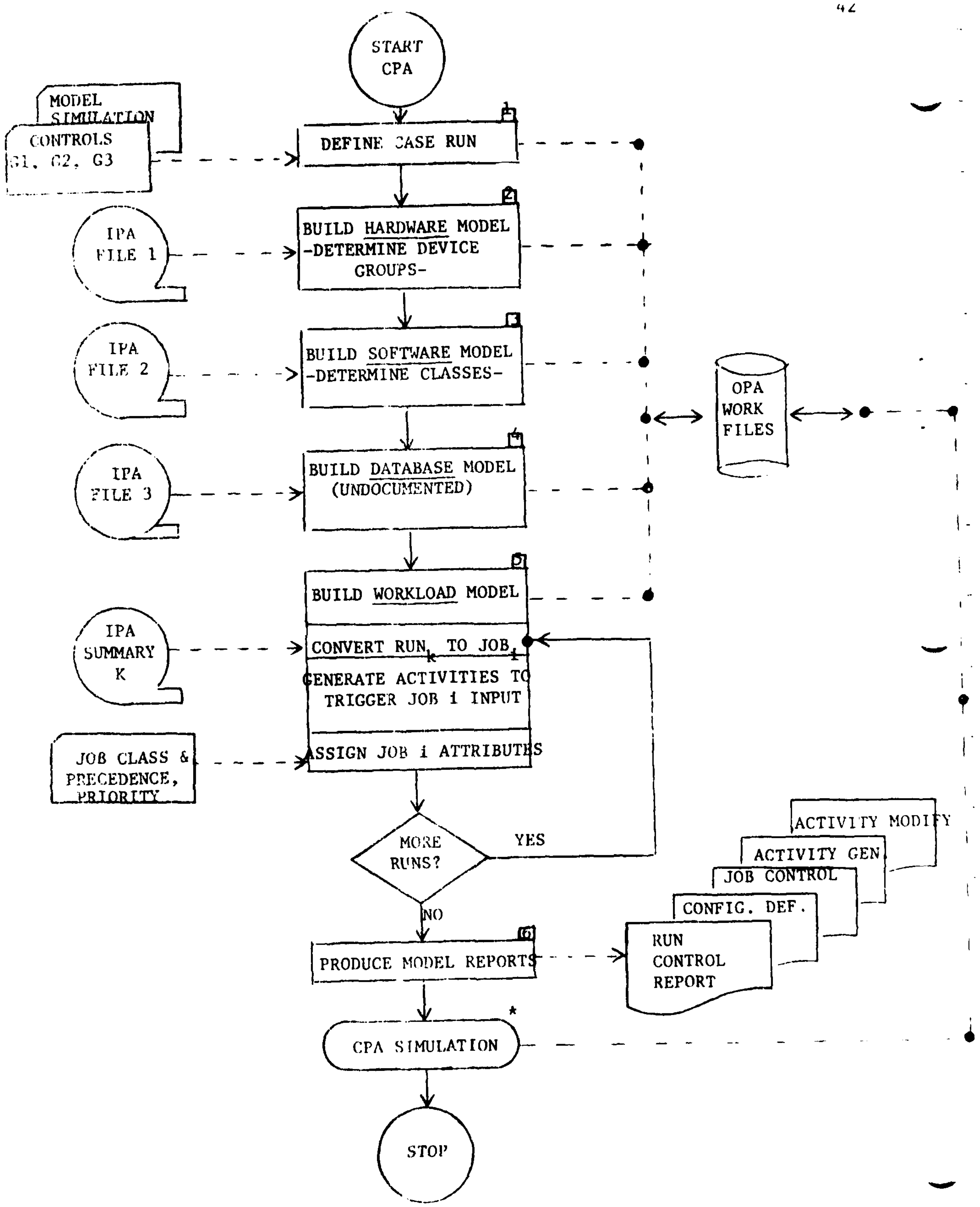
4.3 THE CONCURRENT PROCESS ANALYZER (CPA)

The CPA phase of CASE is employed to analyze the concurrent processing of jobs. This module uses outputs generated from the IPA module, along with additional user controls, to effect the desired multiprogramming simulation. A run schedule is determined, contingent upon user-defined timing constraints. The jobs are then simulated as concurrent processors. Output includes CPA-assigned run priorities, critical activity analysis of system modules, on/off times for all jobs, and utilization statistics.

As was true for IPA, the CASE CPA module can also be run iteratively until the modeler has satisfied his requirements. The CPA simulator can be separated into two phases: model building and model execution. These two phases collectively define the CPA simulator, and are considered separately herein to keep the discussion clear.

4.3.1 THE CPA MODEL BUILDING PHASE

The model building phase constructs characterizations of the hardware, executive software to include the operating system, the user-defined (and IPA-amplified) workload, and database. Figure 4-3 illustrates the basic steps of the CPA model building phase. When this phase is complete, CPA automatically enters the simulation phase, which we describe following this discussion. Amplification of the 6 basic steps of the CASE CPA model building phase (Figure 4-3) follows.


START CPA
MODEL SIMULATION CONTROLS G1, G2, G3
DEFINE CASE RUN
1
IPA FILE 1
BUILD HARDWARE MODEL -DETERMINE DEVICE GROUPS-
2
IPA FILE 2
BUILD SOFTWARE MODEL -DETERMINE CLASSES-
3
OPA WORK FILES
IPA FILE 3
BUILD DATABASE MODEL (UNDOCUMENTED)
4
BUILD WORKLOAD MODEL
5
IPA SUMMARY K
CONVERT RUN$_k$ TO JOB$_i$
GENERATE ACTIVITIES TO TRIGGER JOB i INPUT
JOB CLASS & PRECEDENCE, PRIORITY
ASSIGN JOB i ATTRIBUTES
MORE RUNS?
YES
NO
PRODUCE MODEL REPORTS
6
ACTIVITY MODIFY
ACTIVITY GEN
JOB CONTROL
CONFIG. DEF.
RUN CONTROL REPORT
CPA SIMULATION
*
STOP

* See Phase 2 Flowchart

FIGURE 4-3 CASE/CPA Model: Phase 1

1. The CASE CPA run to be executed is defined by the modeler input describing report selection, file assignments, and simulation parameters defining operating hours, maximum CPU loading, number of priority levels, and page reference numbers.

2. The hardware model is built using the IPA hardware output data which describes the device and its characteristics and its interconnections. CPA builds groups of all devices having common function, manufacturer, and model number. Each group is assigned a unique id.

3. The operating system model is also constructed from IPA output. CPA determines region sizes (both shareable and non-shareable) for run simulations to simulate memory allocation regions.

*4. The database or files model construction is completely undocumented and thus cannot be further described.

5. The workload model is the crux of the CPA model to be constructed. It reads the output of each IPA run (declaring space and time requirements per device) and includes it in the CPA run table if the user wishes it to be included in the CPA analysis. Runs are combined (by user input definitions at this time) to become jobs. The Execution times per month for each job determines the generation of job submissions to the simulated system. These will represent the activities to be processed in time by CPA. Further modeler input can define predecessor/successor relations between certain jobs that CPA must honor when simulating scheduling and execution of these jobs.

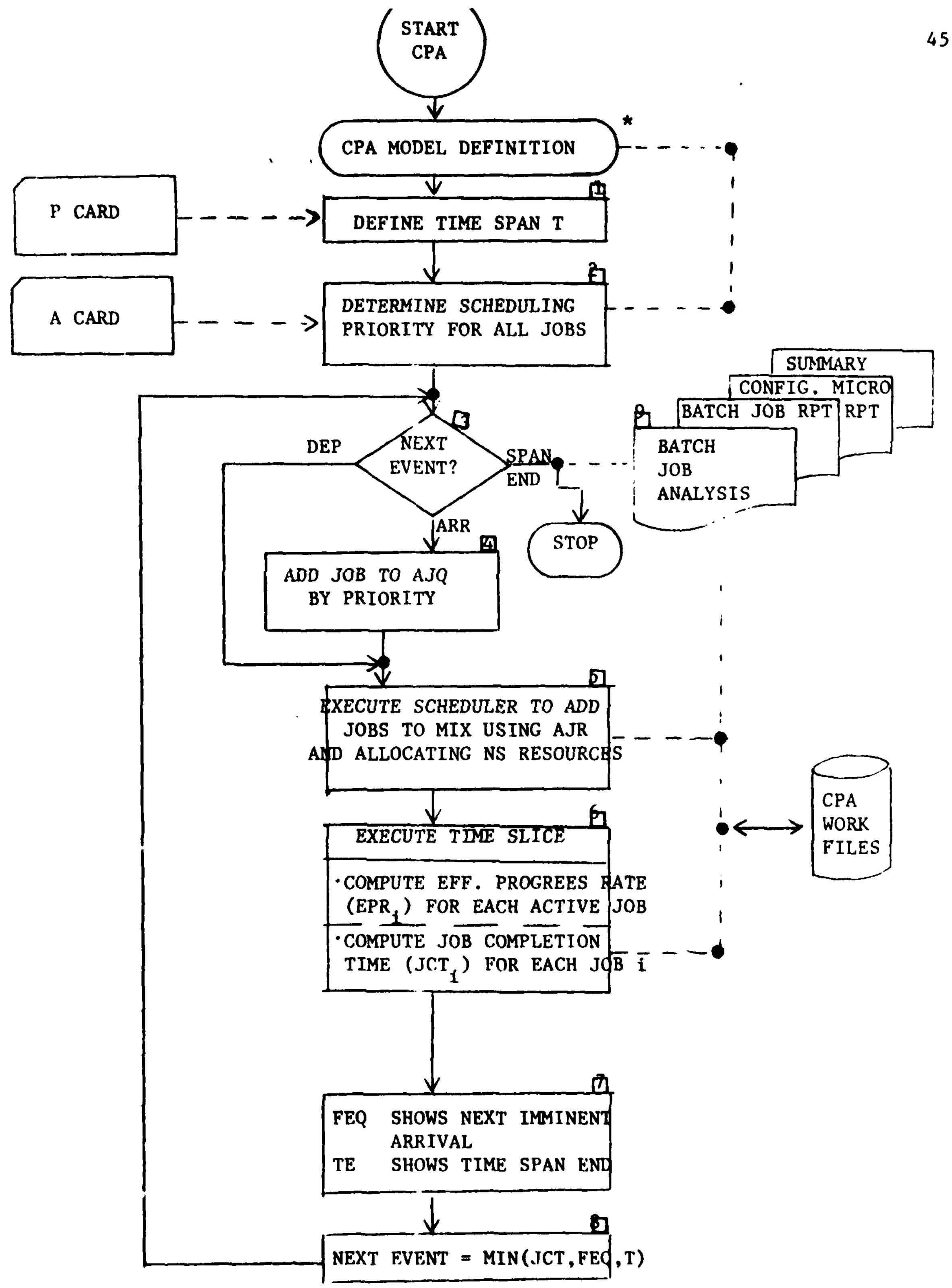
6. The CPA model building phase concludes with the generation of output documenting the model built. These are further described in the CPA output section of this chapter.

At the conclusion of the model building phase, the simulation model is completely defined and ready to execute. Control passes automatically to the simulation phase of CPA to complete the simulation.

4.3.2 THE CPA SIMULATION PHASE

The simulation phase simulates the concurrent execution of the modeler-defined workload on the defined system as constructed by the model building phase of CPA. This is mainly a probabilistic (as opposed to discrete event) simulation. The only events modeled are program starts and stops; these determine the varying time slices for which statistics are generated. Figure 4-4 illustrates a high-level flowchart constructed by the author to describe CPA simulation. The basic eight steps shown are augmented by the accompanying discussion.

1. The time span to be simulated is modeler-defined and used by the simulator to create an event which, when encountered in simulated time, will halt the simulation.
2. The scheduling priority for each job is determined by executing an topological sort, which ensures the presentation of all successor/predecessor relationships between jobs. The modeler may further define time deadlines for certain jobs which will also be taken into consideration at this time.
3. Time is incremented by the simulator to the time of the next imminent event; execution continues dependent upon the event type; time completed, job arrival, or job departure.
4. Every job arrival activates the job scheduler (defined in the CASE library) which places the job in the proper place in the Available Job Queue (AJQ) by its priority. These jobs are waiting to be executed but have not been initiated.



START CPA
CPA MODEL DEFINITION
*
P CARD
DEFINE TIME SPAN T
1
A CARD
DETERMINE SCHEDULING PRIORITY FOR ALL JOBS
2
SUMMARY
CONFIG. MICRO RPT
BATCH JOB RPT
9
BATCH JOB ANALYSIS
3
DEP
NEXT EVENT?
SPAN END
STOP
ARR
4
ADD JOB TO AJQ BY PRIORITY
5
EXECUTE SCHEDULER TO ADD JOBS TO MIX USING AJR AND ALLOCATING NS RESOURCES
CPA WORK FILES
6
EXECUTE TIME SLICE
·COMPUTE EFF. PROGREES RATE (EPR_i) FOR EACH ACTIVE JOB
·COMPUTE JOB COMPLETION TIME (JCT_i) FOR EACH JOB i
7
FEQ SHOWS NEXT IMMINENT ARRIVAL
TE SHOWS TIME SPAN END
8
NEXT EVENT = MIN(JCT,FEQ,T)

*see Phase 1 Flowchart

FIGURE 4-4. CASE/CPA Simulation: Phase 2

5. The job scheduler (defined in the CASE library) is activated every time a job arrival or departure occurs. It examines the AJQ to determine which jobs can be activated to enter the mix of active (executing) jobs. This is based upon job placement in the AJQ and the availability of non-shareable resources required by the job. The initiation (defined in the CASE library) simulates job initiation whenever a new job is to enter the job mix.
6. Every job entry/departure ends a time slice. The CASE micro-queueing algorithm computes the Effective Program Rate (EPR) for each active job. $EPR_i = 0$ implies that job i can run as fast as if it were the only job executing (this $TIME_i$ was calculated by IPA). EPR = 50% implies the job will take twice as long to execute in this job mix as it would were it executed independently. The EPR for each job is normally greater than zero due to priority and resource contention for shared resources. Given the CLOCK as the current simulated time of day, then Job Completion Time for each job i (JCT_i) is predicted as follows:

$$JCT_i = \frac{TIME_i}{EPR_i} - PST_i + CLOCK \tag{1}$$

where PST_i is the Previous Slice Time already accumulated for job i.
7. The first element of the future events queue (FEQ) denotes the next imminent job arrival to the system. TE denotes the time span end, as computed from step 1.
8. The simulator increments the CLOCK to the time of the next event, which is either the smallest JCT_i (a job departure) or the first event in the FEQ (a job arrival) or TE, simulation halt time, whichever is smaller.

9. At simulation completion time reports are generated to document the simulation. These include the batch job analysis, batch job report, configuration microqueueing report, and summary report.

At the conclusion of this CPA simulation the modeler examines the outputs generated to determine if the simulation performed as desired, or if CPA should be re-run with input changes. IPA need not be re-executed unless changes to the initial job stream definition are desired.

4.4 CASE INPUT

Now that we have overviewed the CASE simulator, we can, with some better understanding, focus upon the inputs available to the modeler to define and characterize a given CASE simulation study. The one area of CASE documentation that is defined in depth in the CASE manual is the description of CASE inputs. Unfrotunately, it is not discussed adequately with regard to helping the user understand the implications of his input definitions (the "why's"), but the "how to" portion is excellent and hence will not be completely detailed in this report.

As the CASE system uses a "black box" approach to simulation, the entire flexibility allowed the user is restricted to definition of CASE-provided parameters. All input to CASE is strictly formatted and poorly documented (on the input itself) so errors are easy to make and overlook. We separate the input discussion into two sections, to IPA and CPA, respectively, in an effort to briefly outline the options and capabilities provided the modeler by CASE.

4.4.1 IPA Inputs

Modeler inputs to IPA can be discussed in the five classes shown in Figure 4-2: control information, configuration items, job file descriptions, transmit definitions, and run descriptions. Lastly we discuss the CASE library, as it is used to amplify the user definitions and references.

Control Information

The first eight records input to IPA are controls to be imposed on the IPA module. These controls include: 1) user selection of IPA output reports; 2) time units for simulation reports and frequency for runs and reports; 3) disc/drum organization; 4) general sort characterization; 5) maximum size of master and temporary files; 6) identification of removeable packs.

Configuration Items

The hardware/executive software environment to be simulated is referenced in this portion of the IPA input. CASE module codes must be used to link these references to further definitions in the CASE library. The quantity and connections of each unit is specified. These definitions can be modified by user $LIF cards to alter existing CASE device definitions or to add new devices not in the current CASE library. However, these new definitions must be defined strictly in the terms of the other CASE definitions. E.g., no new parameters can be added to characterize devices: only CASE-defined parameters may be utilized.

File Definitions

CASE works with both master (permanent) and temporary files. Each file defined includes the blocking factor, average record length, magnetic tape density or disc seek time, organization (sequential, random, or indexed sequential), and device assignment. This set of definitions can be easily modified for other IPA runs without altering these initial definitions. IA Organization records can be defined to simulate other file organizations. We will discuss these IA tables in a later chapter, as they provide the vehicle for database modeling.

Transmit Definitions

Transmit definitions are available to the modeler to simulate remote data transmissions of a real-time nature. These are simply passed on by IPA, with their associated timing characteristics, to CPA where they will represent exogenous events just like run submissions. Transmit data includes terminal speed, probability of selection, and think time (inter-record time).

Run Definitions

CASE runs are classified as either general, sort, or translation. The user describes the run as an update, report, or extract, if appropriate; the source language (FORTRAN, ASSEMBLER, COBOL, RPG, PL/1, or ALGOL); run frequency, file references; type if refereshable, serially reusable, or reentrant. File usage is characterized by type (I/O/Update), disposition (NEW, KEEP, DROP), percent of file used, trigger file, buffering mode (SIMPLE, EXCHANGE), and internal buffer storage requirements. Output requirements such as print format, number of lines, copies, etc., are also noted.

The most critical part of run characterization is a description of the actual processing to be simulated. CASE does an excellent job of this by providing process operations to be used by the modeler. These include the ADD, SUB, CMP, BRCH, EDIT, MOVE, and LOOK-UP operations. The user specifies the number and probability of executing such operations for each file access. If a more global description is desired (maybe the program being simulated does not exist) then CASE provides general operations which represent weighted averages of the discrete operations. These include mixes to describe Housekeeping, Decimal Math, Business, Update, and Scientific Applications. This provides the modeler with a good vehicle for job description, at either the micro or macro level.

The CASE Library

Modeler inputs to IPA are augmented by the CASE library. It contains data pertaining to selected manufacturers' hardware devices and executive software. Thus the modeler need only reference this data and CASE automatically completes the definition.

Of particular interest is the CASE library definition of the database manager. It defines the simulated dynamic buffer pool memory size and the database manager's resident core requirements. It further defines the average number of instructions (CPU cycles) to perform the database operations OPEN/CLOSE, RETRIEVE/REPLACE, and INSERT.

4.4.2 CPA INPUTS

Modeler inputs to CPA can be discussed in the four classes noted in Figures 4-3 and 4-4 collectively. These inputs provide simulation flexibility in user control of the simulation execution, job class assignments with priority and precedence, simulation time span, and job descriptions. The following paragraphs highlight these four areas of CPA input.

CPA Simulation Controls

The modeler selects the CPA reports desired, to include queue time delays and time-slice reports and determines if CPA batch job scheduling is to be performed or not. He further defines the file assignments for the IPA-generated files. Lastly controls defining system operating times, number of priority levels, random number generator seed, and CPU loading limit are specified.

CPA Workload Selection

Run selection controls provide the modeler the tool with which to direct CPA to consider only those specified runs (from the IPA output) to be analyzed by CPA. Job description controls are used to convert selected runs into CPA batch or real-time jobs. A job is a collection of run or control or transmit steps. The run and transmit steps were analyzed by IPA and are combined with user-defined control steps to define a CPA job. All selected runs not referenced by user job descriptions are automatically simulated as separate batch jobs by CPA. Control steps (LOOP, BEGR, DELA, BEGT, etc.) enable the modeler to request CPA recording of wait times and response times and to define job step sequencing for simulation.

Job Scheduling

The user may indicate predecessor/successor constraints for the simulated job stream via the JC controls. He may also link batch jobs, define page fault rates, and specify virtual memory usage. Furthermore, with each defined job the modeler uses A cards to define activities (events) for job submission generation and regeneration. Inter-arrival times may be functionally defined by inputting discrete point graphs to CPA.

Period Definition

The modeler defines the simulation time span, and the units for simulation: hours, minutes, etc. He also requests either analytical or event simulation to be carried out. CPA simulation of job arrivals and departures is always discrete; it is the time-slice analysis which is probabilistic.

4.5 CASE OUTPUTS

Execution of a CASE simulation results in the production of several reports. These are either documentary in nature or data for the modeler/analyst. As the documentary reports are not of significance to this study, we restrict the discussion to the reports for the CASE analyst/modeler. Our analysis of CASE inputs showed the flexibility of CASE for user-definition of simulation models. The output of these models is considered most important, as the utility and value of CASE lies totally in the value of these reports for modeler analysis and conclusions. While CASE provides a variety of reports (as shown next) they are not well documented with regard to helping the modeler understand the I/O relations; yet this is what simulation is all about.

4.5.1 IPA REPORTS

No less than 25 reports can be requested by the modeler and produced by IPA. Of these twenty-five, 16 are documentary and 9 are useful for simulation analysis. Discussion is focused upon run processing reports, run detail reports, and the summary reports.

Run Processing Activity Detail Report

This report shows each processor operation associated with a given job. Detailed for each process are: probable number of times executed, associated file, number of operations in the user-defined process, memory requirement of process, and execution time. The probable number of times a process is executed is defined by CASE as the file size times its percent utilization times the processes' probability of execution. This fact emphasizes the fact that CASE utilizes probabilistic as opposed to discrete event simulation techniques.

Run Detail Report

This report shows the time requirements (channel, run delay, and unit) for I/O on each file referenced. By program, I/O, and system it shows memory and processor requirements. I/O overhead times and total times (delay, execution, charge, and elapsed) complete this report.

Summary Reports

The File Summary details process wait time and file utilization time. All transmits are summarized by name time for a single execution. Run summary shows processor, charge, and elapsed time for each run. Unit Utilization shows, for each unit, the total delay time, total usage time, and total allocation in characters. Channel Utilization shows busy and utilization times per channel.

4.5.2 CPA REPORTS

A total of 33 reports can emanate from a CPA execution if desired by the modeler. Fourteen reports are produced at the end of the CPA model building stage; two at the start of the simulation stage; two after each time slice; fourteen at the end of the simulation.

For brevity's sake, we will comment only on the reports produced after each time slice. All reports are described in the CASE User Manual.

Concurrent Processing Slice Reports

This report details the jobs executed during each time slice. It shows the effective progress rate of each job; which jobs started; which jobs completed; the cause of delay if not 100% effective progress rate.

End Simulation Reports

At the completion of CPA, reports are produced to characterize the following: partially completed jobs, available job queue, real time activity, real time jobs, real time control site response time, real time macroqueueing, memory contention, batch job details, batch job summary, configuration macroqueueing, configuration microqueueing, communication network queueing, system idle times, and concurrent processing summary.

5. DIMUI OVERVIEW

The Database Implementation Model Using Industion (DIMUI) is a special purpose simulation facility designed to enable one to model computer systems, with emphasis upon database input/output aspects [7]. Its development has been under the direction of Dr. Allen Reiter, Associate Professor, Computer Science at the TECHNION - Israel Institute of Technology. During the past several years, the DIMUI project has been supported by the United States Army (AIRMICS), and has grown from an initial system design to a large system simulator which has significant capabilities in the areas of database modeling. Research continues with TECHNION Ph.D students to expand and complete the DIMUI implementation under Reiter's direction.

The DIMUI simulator includes modeler definition of data structure and data base task descriptions (see Figure 5-1); hence it is not quite so musch a "black box" methodology as is CASE. Yet it is a hgihly parameterized simulator. DIMUI is oriented to database analysis from the operating system level interface. This enables both static and dynamic issues in multiprogramming/multitasking environments to be adequately represented and explored through simulation of database activities.

DIMUI consists of a number of interrelated FORTRAN programs which can be parameterized and executed to simulate a multiprogrammed computer engaged in database management tasks; it includes associated processors to translate high-level user data definitional forms into lower forms suitable for simulation. While the major part of DIMUI is not dependent on any particular DBMS or its implementation, different DBMS dependent modules must be added to obtain a performance model of a specific DBMS. Thus the models DIMUI/ISAM, DIMUI/IDMS, and others have been created to simulate specific DBMS systems.

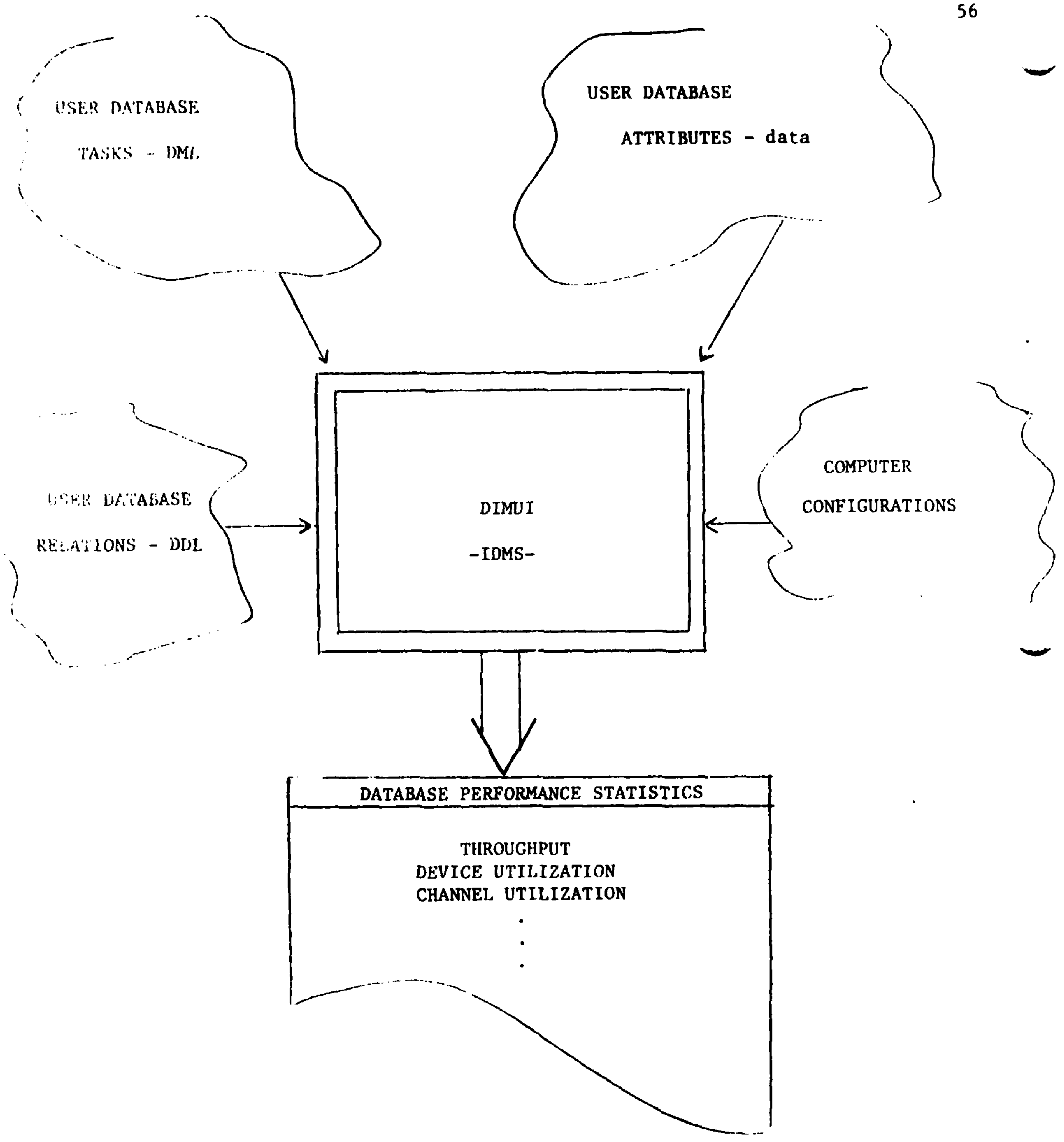


FIGURE 5-1. The DIMUI/IDMS Simulator

The Reiter Model is of the inductive kind: a model of a computer system is synthesized from basic building blocks. It addresses computer systems whose chief function is manipulating large amounts of data. DIMUI assumes I/O bound systems, therefore devoting relatively little attention to the CPU.

5.1 THE DIMUI METHODOLOGY

DIMUI was designed to be an evaluative tool for database systems. As such, it focuses upon the interaction of the database management system (DBMS) with the operating system in servicing user job functions. Each user I/O request goes through the operating system to the DBMS to the IOCS to carry out the actual I/O required. It is Reiter's belief that the determining factor in database performance is what occurs at the device level. Hence DIMUI was designed to very carefully model the I/O resulting from DBMS activity in microscopic detail while modeling the rest of the computer system in a macroscopic manner.

Figure 5-2 illustrates the DIMUI view of an information system and how its database capabilities can be simulated. Jobs must be defined (stage 1) that interact with the DBMS. These definitions will include (and the simulation will focus upon) DDL and DML statements that provide a high level, logical definition of the task to perform and the user's view of the database. Each job creates a sequence of DML commands to be serviced by the DBMS; this is the input/interaction of stage 1 and stage 2.

At the DBMS modeling level (stage 2 of Figure 5-2) the DBMS must be modeled. The DBMS task is to translate the logical data requests (stage 1 DML statements) into physical I/O requests to be carried out by the computer system simulator (stage 3). This cannot be accomplished without knowledge

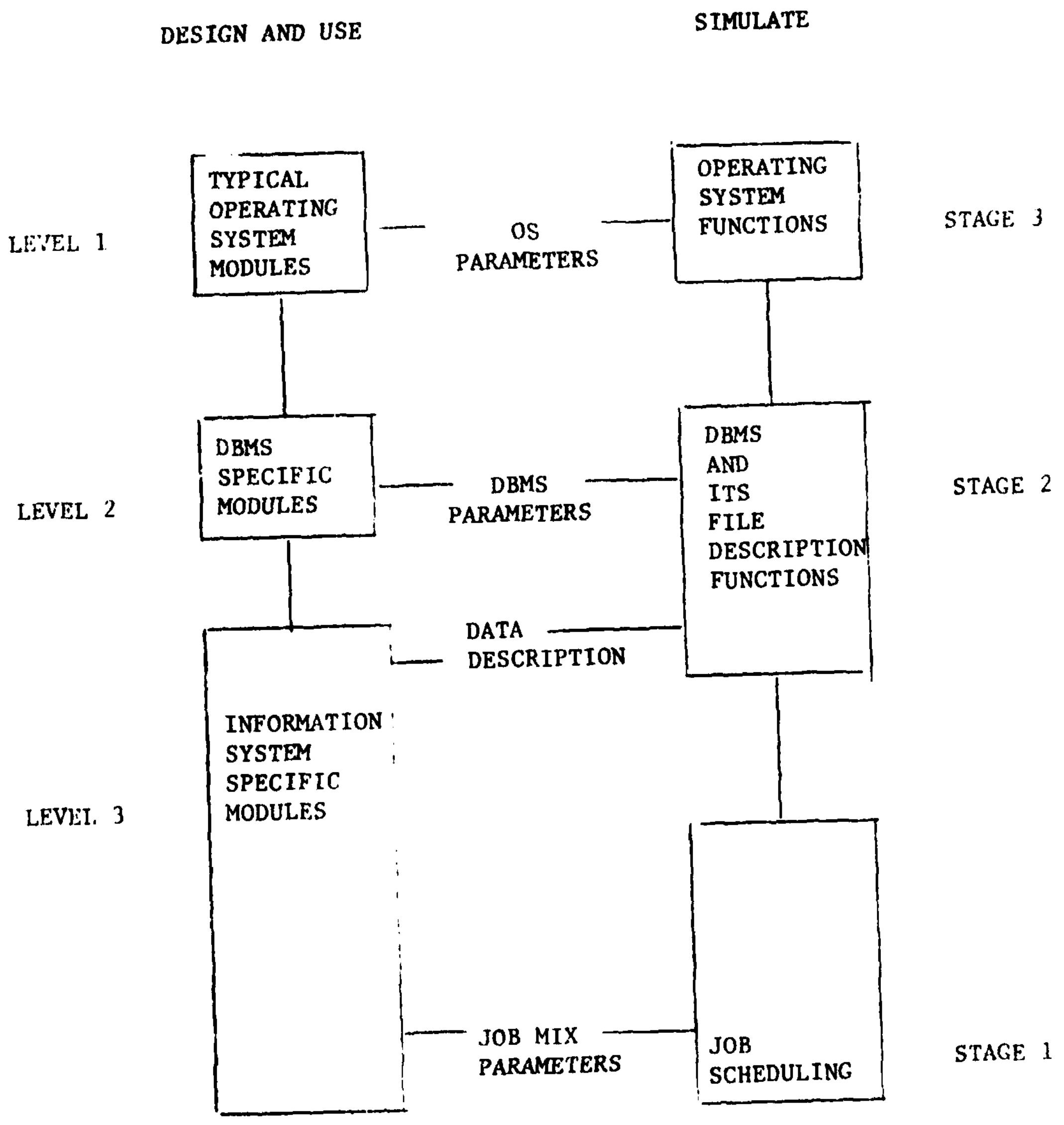


FIGURE 5-2. The DIMUI Methodology

of the logical database defined in the stage 1 DDL statements and the specific physical implementation of the DBMS being modeled. Also required is the capability to "instantiate" the portion of the database under reference to simulate task performance.

The final phase (stage 3) is to carry out the physical I/O requests in a multiprogrammed computer enviornment to obtain performance measures. This system simulator considers device definitions and their channel connections, space management, resource allocation, job and task scheduling.

Several simulators exist (none non-trivial) that carry out the basic task of stage 3 of the Reiter model. (Note, however, that CPU utilization is macro-modeled in stage 3 as emphasis in on I/O.) The contribution in the DIMUI methodology lies principally in devising the input for stage 3: stages 1 and 2. Therein the logical-physical binding and database representation is effected, and this is indeed a difficult task. Furthermore, it is different for each specific DBMS. The overall DIMUI approach, nonetheless, is a <u>general</u> one, one that can be tailored (albeit non-trivial) to any DBMS.

The crux of the problem is translation of user logical requests to physical requests by somehow instantiating the database and simulating data traversal. The DIMUI system revolves entirely around the solution generated: a generalized data structure definition with accompanying traversal commands [8]. Reiter has developed a basic set of data structures that, although logical, can be related to physical page placement. The database is considered to be a network of nodes, with nodes being connected by either points, adjacency in same block, adjacency in next adjacent block, or disconnected.

Reiter has demonstrated, at least theoretically, that this primitive set of data structure constructs is sufficient to characterize the physical realization of any DBMS. He further designed a small language, SIDBL to effect data traversal. Hence the different DIMUI simulators can be created by changing the interface between the host language DML statements and the appropriate SIDBL commands to traverse the instantiated database portion to effect I/O requests when appropriate.

Thus DIMUI analyzes database activity at a very primitive level (which includes directory searching, for instance, which is often ignored). This gives it the capability to simulate this activity in a microscopic manner and provide an accurate reflection of I/O requests. Below this one must simulate actual I/O (stage 3) and above this user tasks must be transformed into SIDBL tasks (stage 1).

Hence DIMUI emerges as a three stage system for translation, synthesis, and execution of the user tasks for performance evaluation. The following three sections, in conjunction with Figure 5-3, serve to more fully describe the DIMUI system.

5.2 STAGE 1: TRANSLATE

DIMUI must make two types of translations in stage 1, as shown in Figure 5-3. Modeler input is provided in the form of DDL statements to define the user view of the database, and user definition of tasks using DML statements. When DIMUI is a completely finished product, the modeler input for database traversal will not deviate much at all from actual DML-oriented application programs that would run on the DBMS being simulated.

The DIMUI Data Description Translator is completed, and enables one to input schema definitions in the host DDL, which are translated into the primitive connected node data structure supporting SIDBL. This frees the user from being required to know the underlying SIDBL network that supports the schema, and enables him to use a language quite close to the DDL of the simulated DBMS.

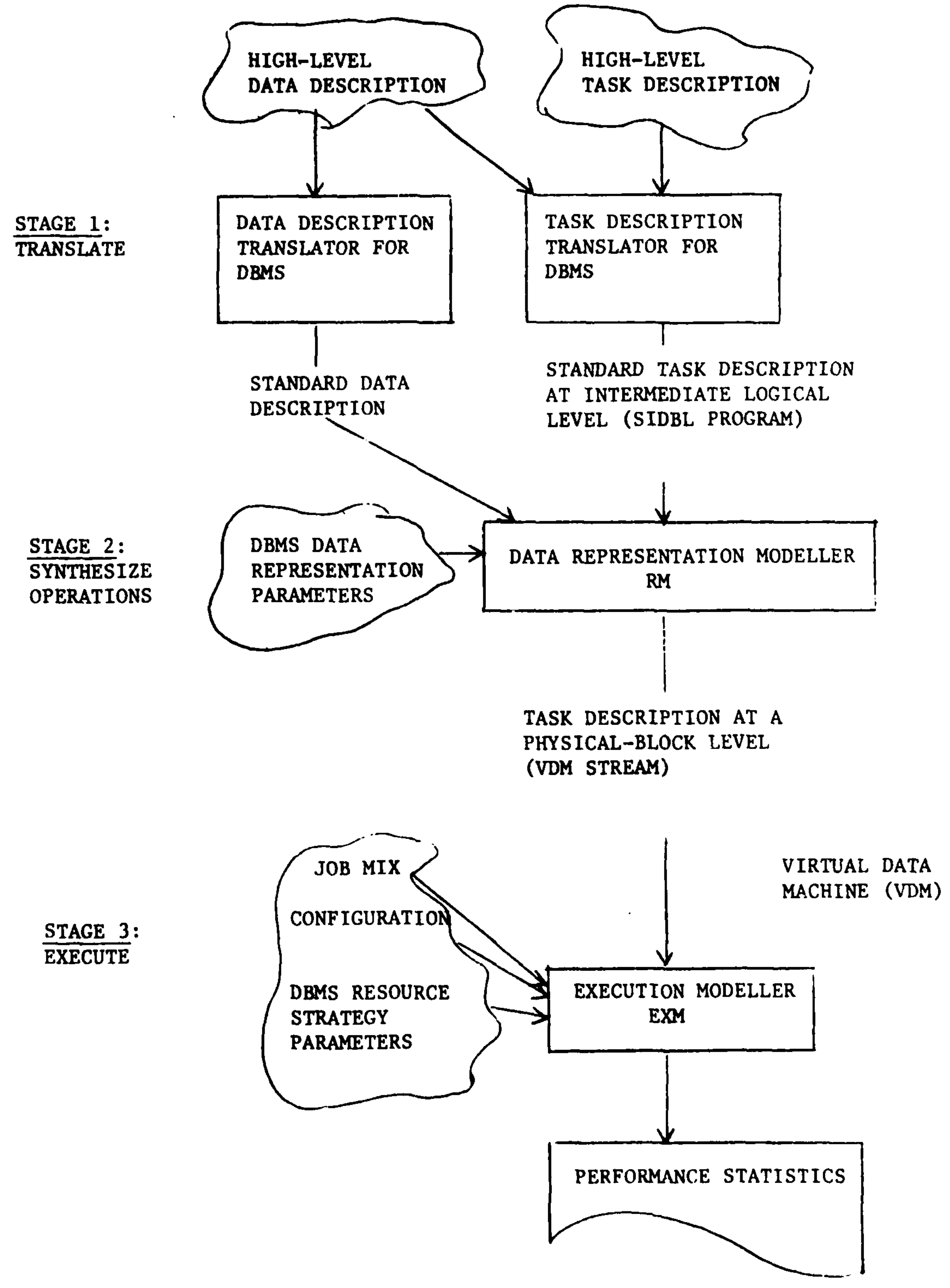

HIGH-LEVEL DATA DESCRIPTION
HIGH-LEVEL TASK DESCRIPTION
STAGE 1: TRANSLATE
DATA DESCRIPTION TRANSLATOR FOR DBMS
TASK DESCRIPTION TRANSLATOR FOR DBMS
STANDARD DATA DESCRIPTION
STANDARD TASK DESCRIPTION AT INTERMEDIATE LOGICAL LEVEL (SIDBL PROGRAM)
STAGE 2: SYNTHESIZE OPERATIONS
DBMS DATA REPRESENTATION PARAMETERS
DATA REPRESENTATION MODELLER RM
TASK DESCRIPTION AT A PHYSICAL-BLOCK LEVEL (VDM STREAM)
VIRTUAL DATA MACHINE (VDM)
STAGE 3: EXECUTE
JOB MIX
CONFIGURATION
DBMS RESOURCE STRATEGY PARAMETERS
EXECUTION MODELLER EXM
PERFORMANCE STATISTICS

FIGURE 5-3. Generalized DIMUI Schematic

This portion of DIMUI represents the latest addition to the simulator. It adds a lot to the modeler; not in power, but in utility and ease of understanding. At present, the Data Description Translator is coded entirely in PL/1 and requires 1024K bytes of storage (a large amount) for execution. Hopefully the language and storage problems will be overcome in later DIMUI enhancements.

The DIMUI Task Description Translator is not yet complete, so the modeler must modify his definition of the application task accordingly. This requires one to describe tasks in FORTRAN, utilizing DIMUI subroutines in place of host DML statements. For IDMS, DIMUI provides subroutines FINDST, FINDDN, FINDNX, and FINDOW to perform the IDMS FIND verbs; SEARCH, STVIA, and STORNX to perform IDMS STORE verbs; MD for IDMS MODFIY/DELETE; INSSET for IDMS connect; DISSET for IDMS remove.

Further, the modeler (and these subroutines) must use stels (stack elements) to explicitly indicate the IDMS implicit concept of currency. DIMUI uses stack pointers during tree traversal to record the path taken so that it can be used to return to nodes above the "current" node under investigation. Given a stel reference, the SIDBL commands LOCATE DOWN/NEXT/UP can be carried out unambiguously to support the IDMS DML operation requested in the user task.

5.3 STAGE 2: SYNTHESIZE

As we pointed out earlier, stage 2 is the crux of the DIMUI model. Herein the I/O effects of data traversal requests are simulated by carrying out the data traversal on a portion of an instance of the database. In order to effect data traversal, one must have an idea of how it is stored. Any DBMS (IDMS in this case) has an orderly manner for data storage. This strategy must be mimicked by DIMUI for true simulation accuracy.

Figure 5-4 illustrates that IDMS stores data by area. Sets are represented minimally as singly linked lists; optional links may be used for prior member and owner record pointers. The DIMUI Representation Modeler must know which pointers exist, for they clearly affect data traversal. For instance, the FIND OWNER command can follow the OWNER pointer if it exists, else the entire set must be traversed.

We also see from Figure 5-4 that data placement is not always random within an area. The VIA property requests record storage physically near the owner record; hence retrieval of that record can be effected "via" the other record. Physical closeness can be advantageous, as page faults should be minimized and thus I/O times enhanced.

Data references cannot be simulated without knowing something about the properties of the database. CALC chains are of unknown length unless the user inputs some overflow statistical probabilities. VIA traversals cannot be effected without knowing something about file density, page size, record size, and set size. Thus the user input to Stage 2 DIMUI enables a portion of the database to be instantiated, albeit statistical. Then DIMUI can simulate how that data was stored, and then carry out the SIDBL navigational commands to get to the desired record occurrence.

It must not be forgotten that our objective is to simulate I/O requests during this database traversal activity. Knowing page size, record size, set size, overflow probabilities, etc. (from modeler input), stage 2 issues commands to the Virtual Data Machine (VDM) of stage 3 which reflect page fetches and stores. The VDM stream for each task is independent of any other user task, as was also true of CASE's IPA module. At the moment, DIMUI limits the size of the VDM file to 2000 entries. It will be the job of stage 3 to determine actual I/O time during execution of the individual VDM streams.

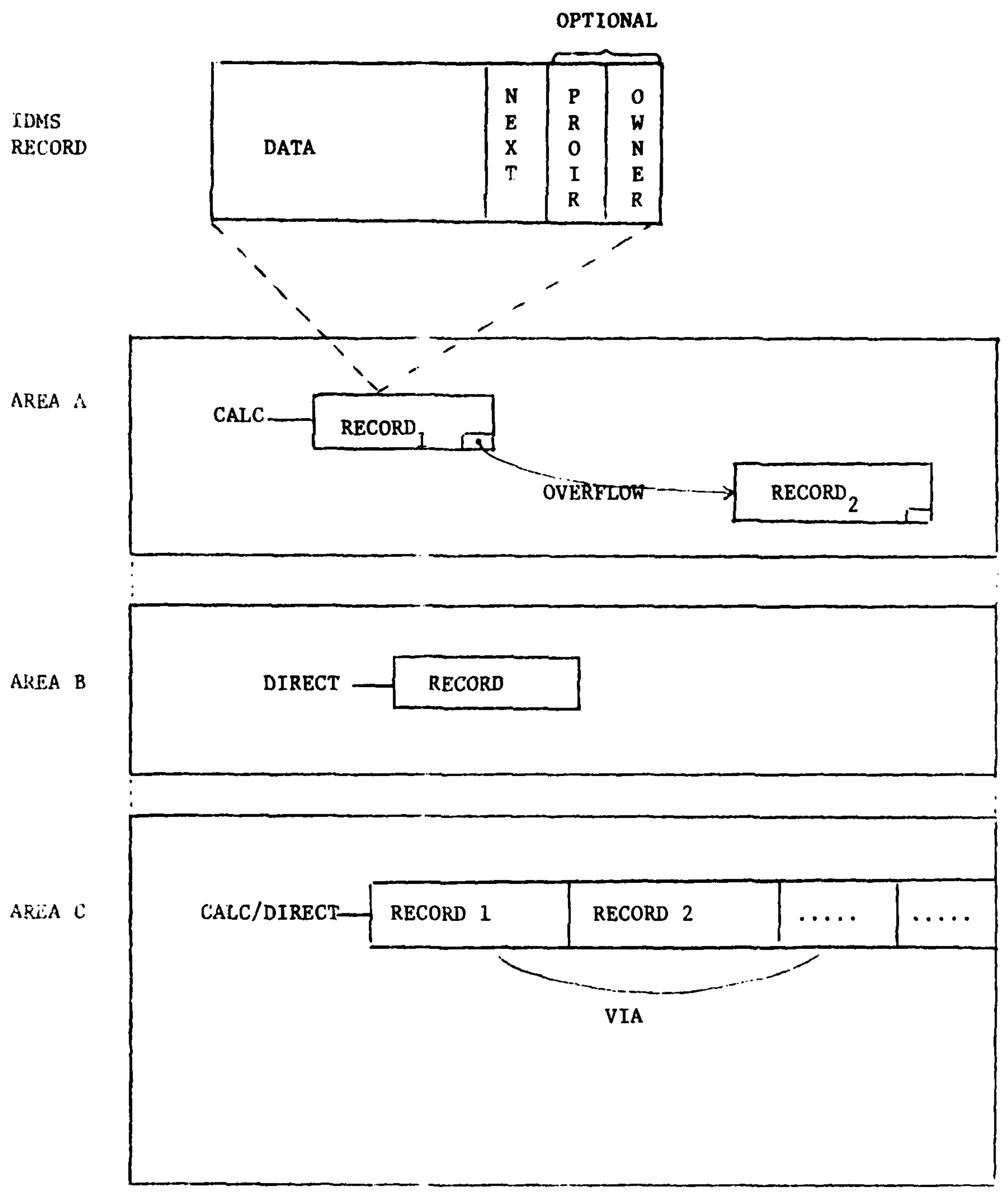


FIGURE 5-4. IDMS Data Storage

5.4 STAGE 3: EXECUTE

Stage 3 of DIMUI is a conventional simulation model of a multi-programmed operating system. It is parameterized to enable the modeler to choose (within pre-defined limits) the type of computer system he wishes for execution of the VDM streams. Figure 5-5 shows Reiter's functional view of the stage 3 DIMUI processor. It performs as does the CASE CPA module, but the emphasis is upon I/O as opposed to CPU-type processes.

The hardware resources represented include models of each input/output unit (primarily disks and tapes), models of main-memory buffers, and models of central and peripheral processors. Each resource model is responsible for keeping track of its status and/or for computing the amount of real time it is busy when initiating a certain operation. The Reiter disk model computes the seek time based on the current rotational position and on the block address and size. Interrupts are simulated by posting a central time-ordered interrupt queue.

In addition to its modelling of hardware resources, EXM is equipped with software resource handlers. The job scheduler is responsible for the activation of one or more tasks. Once a task enters the system, it progresses toward completion as its required resources become available. A task is never concerned explicitly with input/output operations; blocks are read or written at the convenience of the operation system.

Stage 3 of DIMUI is parameterized for modeler-choice of resource management relating to DBMS performance, and is intended to be general enough to model a wide class of operating system and hardware architectures. In the multiprogramming mode, it is the user's responsibility to define which queue a job will enter. The internal DIMUI scheduler and initiator takes over after that.

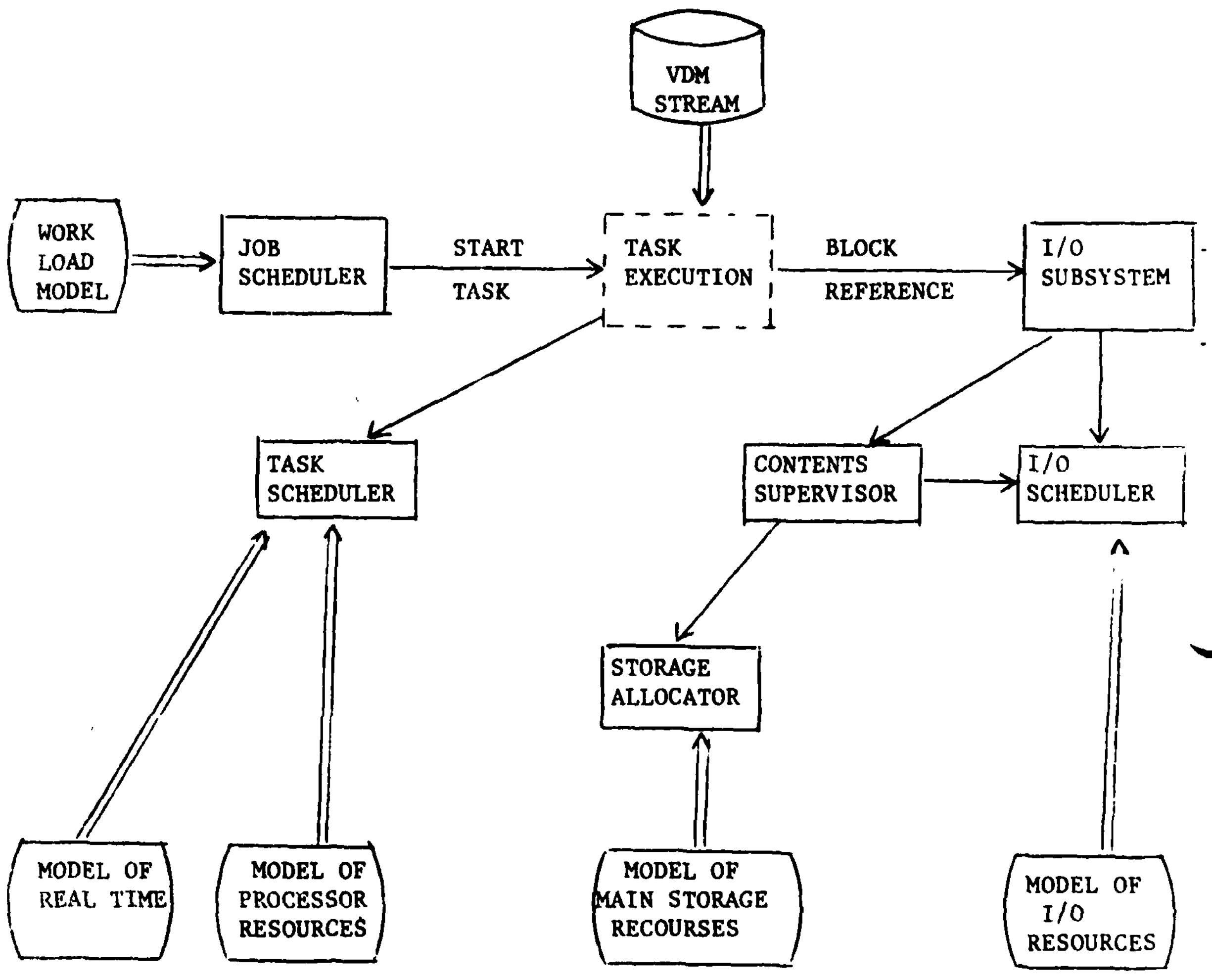


FIGURE 5-5. Functional View of Stage 3

Within a given job mix, each I/O operation is performed when required (not already in buffer, in new block, channel and device are available) and takes into consideration DASD factors such as seek time (rotational delay + arm positioning) and transmission time to determine simulated I/O time. The simulation clock may also be advanced by user command to reflect some sort of record processing time.

The Storage Allocator deals, in particular, with the modeling of several user-chosen buffer management strategies. This is done in great detail, and clearly is an important part of the database I/O activity. All of these strategies modeled must be compatible with the DBMS strategy under investigation. Around 10-15 percent of stage 3 is DBMS-dependent.

5.5 DIMUI OUTPUT STATISTICS

Table 5-1 provides a sample of the output of Stage 3 of the DIMUI model. Outputs of stages 1 and 2 are not really slanted towards the modeler; stage 1 output is simply SIDBL-oriented for stage 2 and offers little or no insight to the user; the VDM stream queued by stage 2 can be used to understand, to a degree, the database traversal resulting from the input task commands.

The user-oriented DIMUI output all emanates from stage 3 execution. As illustrated in Table 5-1, the output of the current DIMUI is scant at best. Here we have the simulation of a single task, with an elapsed time (MSTCLK) ① of 797852 microseconds. This derivation of throughput time is the main DIMUI output; component utilization statistics follow.

Output item ② shows that a single CPU was modeled, and the 16800 microseconds consumed represented but 21.06% of the available CPU computer time. Unit 1, the disc ③ was used 629852 microseconds, had 0 waits for a channel, and 1 disc access requiring no arm positioning for cylinder. Item ④

```
There are 1 work sources.  System open routine is 7.  System close routine
is 7.  The initial jobs are 1

①  MSTCLK=              797852

②  PROC            BUSY          %            IDLE          %

      1          168000       21.06         629852        78.94

           1 RECORDS WRITTEN ONTO LOG FILE.

③  UNIT USAGE  (USEC)  %  OF MSTCLK   WAIT FOR CHANNEL  %  OF MSTCLK  # OF ZERO SEEK.
      1        629852          78.94                  0          0.0           '.
      2             0           0.0                   0          0.0           ʊ
    OUCB                                        QSIZE

④               0          1          2          3          4          5
    1       168000     629852          0          0          0          0
    1       21.6  %    78.94 %    0.0  %    0.0 %     0.0 %     0.0 %        0.(
    2       7797852         0          0          0          0          0
    .                       .          .          .          .          .
    .                       .          .          .          .          .
    .                       .          .          .          .          .
   12       100.0  %     0.0 %     0.0  %    0.0 %     0.0 %     0.0 %

⑤  CHANNEL USAGE  (USEC) % OF MSTCLK
            1         200902       25.18

⑥  I/O SUMMARY.  BLOCKS TRANSFERRED BY TCB.
    NREC( 1)        0  NREC( 2)     7  NREC( 3)        0 NREC( 4)       0  NREC( 5)    (

⑦  EVENT COUNT
     EVT( 1)=     0.0   EVT( 2)=         0.0  EVT( 3)=    0.0   EVT( 4)=  0.0  EVT( 5)= 0.0

⑧  TOTAL EVENTS=          0.0 THROUGHPUT=        0.0  RECORDS/SEC
```

TABLE 5-1. DIMU1 Stage 3 Output

relates queue sizes to the output units. We see that unit 1 (the only one in use for this example) had an empty queue 21% of the time and a queue length of 1 the other 79% of the time. Given that this example shows only one job executing, queue lengths above 1 do not occur.

Channel utilization ⑤ for the only one used (tied to the DASD) comprised 25% of total throughput time, and 7 I/O requests were made for this task ⑥. Output lines ⑦ and ⑧ relate to user-defined work units (mislabeled events) which were not utilized in this example. This enables the modeler, in a macro-like manner, to emphasize protions of task processing to obtain a final weighted, average throughput time.

6. IPSS OVERVIEW

The Information Processing System Simulator (IPSS) is a special purpose digital simulation design facility designed to enable one to model complex, computer-based information processing systems [9]. Its development has been under the direction of Dr. Thomas DeLutis of the Computer and Information Science Department at the Ohio State University. During the past six years, the IPSS project has been supported by various agencies, such as NSF and AIRMICS and has matured from an initial system design to a large system simulator with its associated methodology of use. The basic IPSS simulator has been recently used successfully to model several large computer systems. Thus it is a usable, although not completed, product today. Nonetheless, research continues to enhance the IPSS to include new concepts and implementations so that its value to a modeler should grow, rather than diminish, in today's ever-changing computer environment.

IPSS is much more than just a language, such as GPSS or SIMSCRIPT; it is a complete design/evaluation tool for the modeler. IPSS defines a methodology by which one can characterize all of the complex elements of an information processing system; IPSS provides language constructs specific to simulating information processing systems in standard FORTRAN; IPSS provides detailed execution-time statistics, under modeler control, for the analysis of an information processing system.

6.1 THE IPSS METHODOLOGY

A critical factor in the potential value of any modeling tool is the viewpoint that must be taken by the modeler in order to use the facility. For instance, in GPSS there exist basic entities such as queues, wait blocks, statistics gathering, etc. But GPSS (and SIMSCRIPT also) is a generalized simulation language with which one can model all the way from ship movement in a harbor to automobile assembly lines to crystal formation. These simulation languages have no basic methodology specific to simulating computer systems. The CASE system has a methodology cast in cement that is basically unknown to the user. The DIMUI system has three-component methodology somewhat known to the user, but like CASE, the user must basically treat it like a "black box" and simply live with it.

IPSS provides a methodology which, although specific to computer systems, is general in nature, and quite flexible. It affords the user a viewpoint from which he can construct a simulation model of any computer system at any level of detail desired. This methodology separates the characterization of a complex information processing system into separate, interconnected components. It gives structure and direction to the user, who has the difficult task of defining just what it is he wishes to model.

Figure 6-1 illustrates the role of the IPSS methodology in the design and simulation of an information processing system. We observe that IPSS provides the modeler a top-down approach to the definition of models. At the top of this figure we denote the loose connection of user system knowledge into a set of data and concepts that describe the Information System. This definition may be concise and complete, showing complete knowledge of the system and processes to be modeled; it may be very vague in all respects; it may be specific with regard to certain aspects and non-specific with regard to other aspects of the information system. It is the role of the

IPSS methodology to enable the modeler, who possesses varying degrees of information about the information system, to construct a model at appropriate levels of detail to satisfy his modeling needs.

The IPSS methodological view is to characterize any information processing system as a collection of four discrete but interacting components. As illustrated in Figure 6-1 these components are: 1) services and inter-service procedures, 2) hardware resources and configurations, 3) data base resources and configuration, and 4) user workload. These four component definitions are sufficient to characterize any information processing system; in particular, computer-based information systems or manual systems can be described.

Services and Inter-Service Procedures

The identification and definition of services and inter-service procedures is an important IPSS contribution, and separates its methodology (and subsequent modeling activities) from other systems such as DIMUI and CASE. A service procedure defines a task - manual or automatic - associating all related actions and times to complete the task. In a computer-based system, this component corresponds to the definition of all system software facilities, to include user application programs, the operating system, and the data management system. Service definitions, of course, are constrained to the level of detail required by the modeler or to the level of knowledge of the modeler. This is true of all four component definitions, and forces the modeler to realize the level of detail appropriate, and to obtain additional information. if required, to properly define each componnet. Note that no computer programming is being performed at this time; we are structuring the model to be defined and isolating user information into the appropriate sets of component knowledge. In any computer programming activity, too

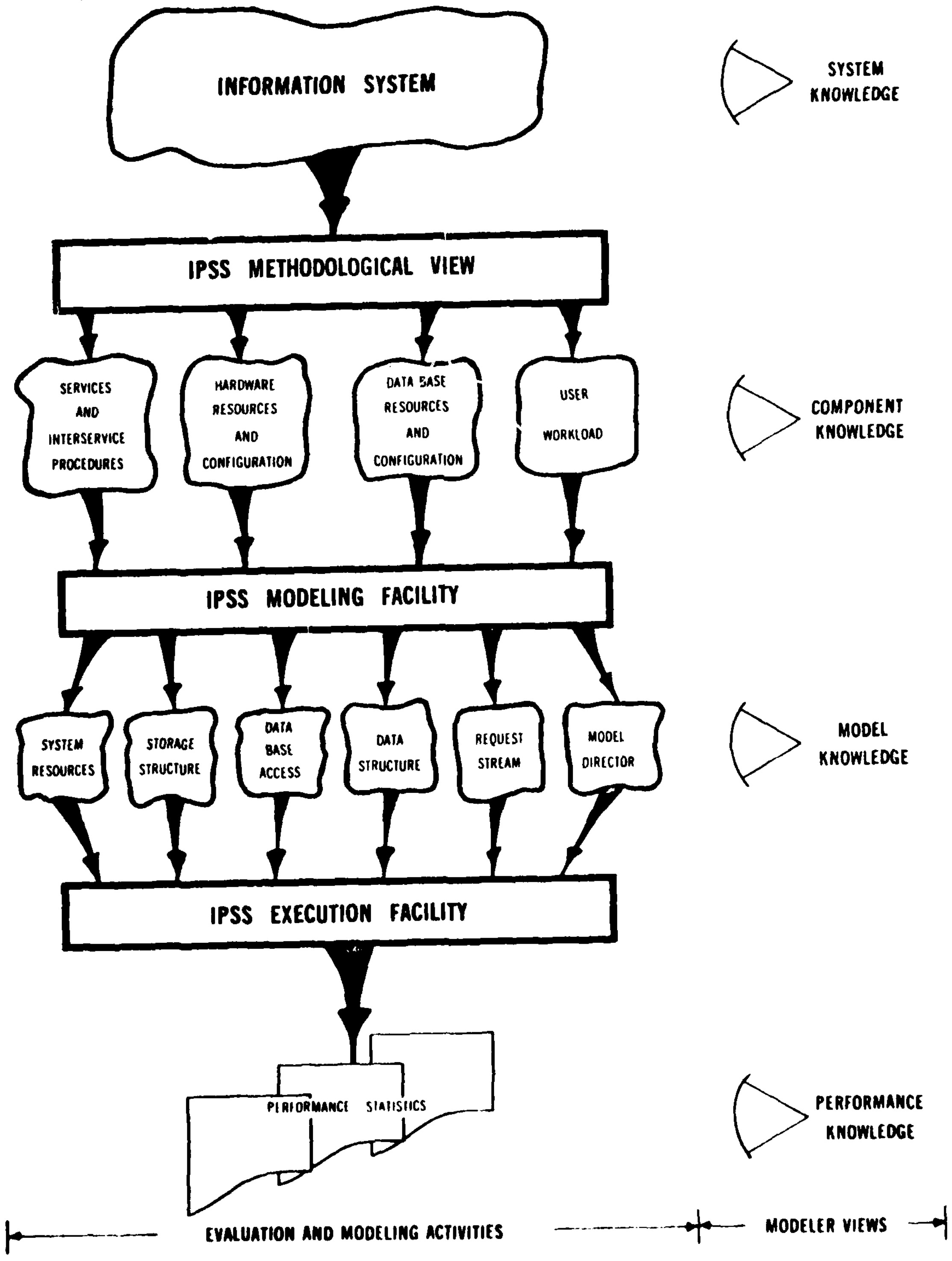


FIGURE 6-1. The IPSS Methodology

much emphasis cannot be placed on structuring the prototype, for correct and appropriate structure can be followed by easy implementation which, by design, should reflect the needs of the modeler.

Hardware Resources and Configuration

The hardware resources and configuration component directly reflects the hardware system to be modeled. This component defines the CPU, immediate storage, tapes, discs, drums, printers, terminals, channel controllers, etc., and all hardware interconnections. Again, the level of detail required is that appropriate to the goals of the modeling activity.

Database Resources and Configuration

The database resources and configuration component defines the logical database of the system to be modeled, to include schemas, file characteristics, database access capabilities, and user data access and data manipulation facilities. This component can reflect a current system with normal non-integrated file management or a future system with fully integrated data management capabilities.

User Workload

Last, but certianly of great importance, is the user workload component. It is here that one characterizes the workload to be placed on the simulated system, to include workload description, timing of inputs, files referenced, etc. This completes the structuring of the user's knowledge of the information system and can be defined functionally or statistically.

A global view of the resultant component is as follows: work (input) to the information processing systems emanates from the user workload and requires certain services. These services may require other services (inter-service procedures) to perform the work required. Whenever database accesses are required, the database resources and configuration component defines and simulates logical data flow while the hardware resources and configuration component simulates the resultant physical data flow. This is the user's view of the information flow process at the conceptual level, structured into components by the IPSS methodology.

6.2 THE IPSS MODELING FACILITY

Given the user's component knowledge as structured by the IPSS methodology, this is transformed by the modeler into model knowledge using the IPSS modeling facility. This portion of IPSS also provides structure and modularity to the model definition, but at a realizable level, as opposed to the conceptual level of component knowledge. The result of this transformation from component to model knowledge is an IPSS-defined simulation model that can be executed by the IPSS execution facility.

There are six model components which comprise the resultant defined model. Given the separation of user knowledge into the four conceptual components defined previously, it is a straight-forward task, conceptually, to define the six IPSS model components which describe system resource, storage structure, database access, data structure, request stream and model director. To actually implement these modules represents a non-trivial, sophisticated effort that requires not only a good understanding of the system to be modeled, but also a complete understanding of how to effectively simulate all of the concepts and interactions of the process to be modeled.

IPSS provides a general simulation language and host enviornment to ease this task for the modeler. The Model Director is supplied for the user, and, in effect, directs the simulation defined by the other five model components. It handles the time clock, and the events queues, and all arrivals and departures from the system during model simulation. CASE and DIMUI effectively pre-define the entire simulation model (especially the system resources model component). This results in a much less understanding about the model; it is the IPSS premise that a modeler cannot effectively use a simulation model that he does not understand.

As a result, IPSS offers a set of language constructs so that the user can, with relative ease, define all important aspects of the simulated activity. Using the IPSS statements, and any additional FORTRAN the user may desire, a FORTRAN model is output from the IPSS translator which can be executed to produce statistics. Additional FORTRAN statements are utilized by the modeler to either add statistics unavailable from IPSS or to model concepts not realized by the IPSS language constructs. In most cases, little additional FORTRAN is required as the IPSS provides a rich set of language constructs with associated statistical capabilities.

The top-down, modular approach provided by the IPSS enables the user to define, using IPSS/FORTRAN statements, five separate model components to characterize the system to be modeled. These are summarized below:

1. System Resources - Contains definitions for all information system resources (hardware and software) and all system tasks (application and operating system). This component forms the basic discrete event digital simulator for the information systems model under investigation. Included in the SYSTEM (system resources component) is the IPSS supplied clockwork mechanism to schedule and control simulated events and to determine when the simulation is to terminate. The clockwork logic is based on the next most immediate event philosophy for controlling discrete event digital simulations.

IPSS statements which ease the modeler's task of defining all of the system resources pertinent to the simulation desired include: Access Mechanism, Area, Buffer Pool, Central Processor, Control Unit, Data Channel, Data Set, Device, Endo Service, Exo Service, I/O Processor, Main Storage, Path, Procedure, Queue, Reference, Semaphore, Task, and Volume statements.

2. Storage Structure - Describes an information system's physical data base storage structure and its space management policies. The STORE (storage structure) component interfaces with the SYSTEM component in three ways. First, it references SYSTEM to obtain Device and Volume facility definitions. Second, it supplies SYSTEM with Data Set facility definitions. Third, it translates secondary storage references specified as a displacement within a data set's logical address space into physical addresses within the secondary storage address space. Prior to a simulation, associations must be specified for the Data Set, Organization Method, Device and Volume facilities. A STORE Organization Method facility can be associated with a multiple number of SYSTEM Data Set facilities. The opposite is true for the Device and Volume facilities. STORE Organization Method facilities are the templates from which the equated SYSTEM Data Set facilities derive their definitions during a simulation. The transfer of definitions between components is accomplished via the execution of the CREATE DATA SET Statement. The space management descriptions in STORE are used to calculate secondary storage addresses dynamically during a simulation based on facility definitions specified in each component and on the changes of these facilities during the course of the simulation.

IPSS statements provided to help the modeler define the Storage Structure Component include: Area, Segment, Organization Method, Extent, Record Type, Device, Procedure, Reference, and Volume.

3. Request Stream - Characterizes the information system's service request stream. It is responsible for the generation of all exogenous events for a model. Whereas SYSTEM contains facilities which characterize the processing requirements for each service offered by an information system, the request stream component (REQUEST) defines the arrival of requests for these services. IPSS converts these times into a composite arrival time stream.

The modeler thus defines exogeneous events, and IPSS eases this task by offering the Exogenous Event statement and the Procedure Statement should the modeler desire to define inter-arrival times functionally.

4. Data Base Access - Contains the definitions of all the resources required by the DBMS. These include the hardware resources of [illegible]fers and user work areas as well as application programs and DBMS software. All DBMS related entity-type facilities are defined within the component. The Data Base Access Component (ACCESS) is similar to the SYSTEM component in that it contains its own simulation clockwork mechanism similar in purpose to the one belonging to the REQUEST component.

IPSS statements particular to the Data Base Access Component include: DML Service, Realm, Schema, Record Origin, Semaphore, Task, and Queue.

5. Data Base Structure - Provides the modeler with a set of facilities which allows the definition of logical data structures and the characterization of relationships among them. This can be applied to a variety of DBMS architectures and application environments. The Data Base Structure component (STRUCTURE) permits the modeler to investigate the effects on system behavior caused by alternate set, record type, and access path definitions. The definitional facilities provided allow the modeler to investigate a wide spectrum of logical data structure organizations and allocation policies.

Within the Data Base Structure Component are IPSS statements to enable the modeler to define the following important database constructs: Realm, Schema, Extent, Record Type, and Set.

6.3 THE IPSS EXECUTION FACILITY

The six IPSS model components discussed in the previous section (MODEL being pre-defined while SYSTEM, STORAGE, REQUEST, ACCESS, and STRUCTURE are user-defined with the aid of IPSS language constructs) comprise the input to the IPSS Execution Facility. It should be understood, however, that this six-component model definition serves not only as necessary input to the IPSS Execution Facility. Of at least equal importance is the fact that the user has now created a documented, readable, understandable definition of the system to be modeled. The fact that this model is explicitly defined at user-determined levels of detail for each model component means that we have a hard copy description of exactly what the modeler wishes to simulate. No implicit assumptions (such as are contained in CASE and DIMUI) exist; hence user verification of the model can be accomplished much more effectively, and the entire modeling effort is at the level of detail desired by the modeler.

The IPSS execution facility carries out the simulation as defined by the six IPSS model components. This execution requires translation of IPSS statements into FORTRAN, link-editing of all required object modules, saving certain user-requested object/source modules in the IPSS library, and executing the resultant load module. Were the user required to define to the computer this multi-step job, a great deal of JCL (machine-dependent job control language) would be necessary. In fact, both CASE and DIMUI require the user to create his own multi-step jobs, a non-trivial, machine-dependent task. The IPSS philosophy is to remove the tedium and complexity of JCL from the user; in fact, the user specifies no JCL whatsoever to execute an IPSS model. Thus IPSS must contain, within its own code, this JCL. We find this within the IPSS Nucleus, which is written in Assembler Language. Hence we find that the IPSS is not completely portable, but only the Nucleus must be re-written to enable execution on another dissimilar machine.

6.4 THE IPSS STATISTICS

IPSS provides a modeler with a number of statistics concerning the behavior of modeler defined entities and IPSS supplied built-in information system services. Many output statistics are provided by IPSS automatically; others can be generated by the modeler's use of IPSS commands to start/end data collection on queues, facilities, services, etc. The IPSS-defined (automatic or modeler invoked) output statistics fall into eight general categories:

1. Operational Statistics,
2. Request Stream Statistics,
3. I/O Activity,
4. Queueing Statistics,
5. Utilization Statistics,
6. Wait Statistics,
7. Service Statistics, and
8. Task/Activity Statistics.

Additionally, the modeler can employ the complete facilities of the FORTRAN language to develop his own statistics. Statistics are printed automatically at the conclusion of each model simulation unless explicitly inhibited. Tables 6-1, 6-2, and 6-3 illustrate some of the IPSS output capabilities.

```
                         INFORMATION PROCESSING SYSTEM SIMULATOR
                                  MODEL SIMULATION PHASE
                                POST SIMULATION STATISTICS

                              PART D - QUEUEING STATISTICS
                                 |
                                 |
FACILITY TYPE = CHN              | FACILITY 1    FACILITY 2    FACILITY 3    FACILITY 4    FACILITY 5    FACILITY 6
                                 |NAME   INDEX  NAME   INDEX  NAME   INDEX  NAME   INDEX  NAME   INDEX  NAME   INDEX
STATISTICS DESCRIPTION           |
                                 |HDSKQ     1   HOBEQ     1   RSPONS    1   TCAMQ     1
                                 |
- - - - - - - - - - - - - - - - -+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                 |
BUSY PERIOD STATISTICS           |
    PER CENT BUSY                |    0.0           0.0           0.5           0.0
    NUMBER                       |   10           290           145            46
    MEAN LENGTH                  |    0.0           0.0         100.0           0.0
    STND DEV OF MEAN LENGTH      |    0.0           0.0           0.0           0.0
                                 |
IDLE PERIOD STATISTICS           |
    PER CENT IDLE                |  100.0         100.0          99.5         100.0
    NUMBER                       |   11           291           146            47
    MEAN LENGTH                  |254545.1       9621.9       19078.7       59574.2
    STND DEV OF MEAN LENGTH      |571445.4      69778.4       97742.3      140279.8
                                 |
QUEUE LENGTH STATISTICS          |
    CURRENT LENGTH               |   0             0             0             0
    MAXIMUM LENGTH               |   1             1             1             1
    MEAN LENGTH                  |    0.5           0.5           0.5           0.5
    STND DEV OF MEAN LENGTH      |    0.5           0.5           0.5           0.5
                                 |
QUEUE ENTRY STATISTICS           |
    NUMBER OF COMPLETED ENTRIES  |   10           290           145            46
    NUMBER OF ZERO TIME ENTRIES  |   10           290             0            46
    PER CENT ZERO TIME ENTRIES   |  100.0         100.0           0.0         100.0
                                 |
TRANSACTION TRANSIT TIME STATISTICS
    OVERALL MEAN TIME            |    0.0           0.0         100.0           0.0
    OVERALL STND DEV OF MEAN TIME|    0.0           0.0           0.0           0.0
    MEAN TIME*                   |    0.0           0.0         100.0           0.0
    STND DEV OF MEAN TIME*       |    0.0           0.0           0.0           0.0
      *(EXCLUDING ZERO TIME SEIZURES)
```

[illegible]

```
                         INFORMATION PROCESSING SYSTEM SIMULATOR
                                 MODEL SIMULATION PHASE
                               POST SIMULATION STATISTICS

                         PART B1 - I/O STATISTICS BY I/O ROUTINE

STATISTICS DESCRIPTION          TOTAL   ZERO   %ZERO    MEAN   STD DEV    *MEAN   *STD DEV
-------------------------------------------------------------------------------------------
SEEK STATISTICS
    NUMBER COMPLETED              19      5     26.3
    TIME IN EXECUTION                                   20.6     12.7      28.0      0.0
    NUMBER OF CYLINDERS CROSSED                          2.3      2.2       3.1      1.9

DATA TRANSFER STATISTICS
    DEVICE TYPE = DASD            19
    NUMBER OF WRITES               0
    TIME IN EXECUTION                                    2.6      0.0
    NUMBER OF CHARACTERS TRANS.                       1042.0      0.0
___________________________________________________________________________________________________

                                PART B2 - I/O STATISTICS BY ACCESS MECHANISM

FACILITY IDENTIFICATION         I/O ROUTINE      NUMBER OF I/O CALLS            ROUTINE TIME
  NAME        INDEX                NAME          TOTAL   ZERO   %ZERO    MEAN   STD DEV   *MEAN   *STD DEV
-----------------------------------------------------------------------------------------------------------
 RP04AA         1                 SEEK            19       5     26.3    20.6    12.7      28.0      0.0

                                  DATA TRANS      19       0      0.0     2.6     0.0       2.6      0.0
___________________________________________________________________________________________________

                                PART B3 - I/O STATISTICS BY DATA SET . FILE

FACILITY IDENTIFICATION         I/O ROUTINE      NUMBER OF I/O CALLS            ROUTINE TIME
  NAME    INDEX   FILE             NAME          TOTAL   ZERO   %ZERO    MEAN   STD DEV   *MEAN   *STD DEV
-----------------------------------------------------------------------------------------------------------
  DS        1       1             DATA TRANS      19       0      0.0     2.6     0.0       2.6      0.0
```

TABLE 6-2. IPSS Statistics

```
                    INFORMATION PROCESSING SYSTEM SIMULATOR

                          MODEL SIMULATION PHASE

                         POST SIMULATION STATISTICS

                        CACHE PAGE FAULT STATISTICS


CACHE SIZE (# PAGES):      5
PAGE SIZE (BYTES):      1024

PAGE WRITE REFERENCES:      17   CACHE MISSES:      0   MISS RATIO: 0.0
PAGE READ REFERENCES:      682   CAHCE MISSES:    662   MISS RATIO: 0.9707

PAGE TOTAL REFERENCES:     699   CAHCE MISSES:    662   MISS RATIO: 0.9471

PAGES REWRITEN BEFORE SPACE REUSED:      17
PAGES REPLACED WITHOUT REWRITING:       645

AVERAGE DISTANCE BETWEEN SUCCESSIVE PAGE REFERENCES:    37.1539
```

TABLE 6-3. IPSS Paging Statistics

7. EVALUATIVE CRITERIA

The second phase of this research activity was to compare and contrast the effectiveness of CASE, DIMUI, and IPSS in modeling database-oriented information processing systems. This task was to be accomplished taking the FEDSIM report [11] comparing ECSS, IPSS, and DIMUI into consideration. This chapter serves as documentation and justification for the evaluative criteria chosen by this author, which are used in the following chapter to form the basis for evaluating CASE, DIMUI, and IPSS.

The FEDSIM report, "Evaulation of DBMS Modeling Approaches", broke its analysis down into three areas: 1) Modeling Capabilities, 2) Modeling Support Facilities, and 3) Program Product Qualities. Within area 1 were some 230 separate concepts assigned to one of the eleven classes: DBMS, OS, TPS, Processor, Storage, I/O, Data, Software, Processes, Workload, and General Purpose Simulacra. Area 2 considered Statistics; Model Behavior Tracing, Synthesis, Verfication, and Validation; Model Initialization, Modification, and Processing; Model Execution and Optimization; Post Simulation Processing. Area 3 considered Acquisition Costs, Maintenance, Portability, Reliability, Documentation and Training.

Clearly this type of evaluation was not appropriate for this research effort because of time constraints, and also to avoid duplication of effort. The end result is that this author has chosen 11 general criteria which he feels are most appropriate to DBMS modeling and which can be adequately justified and discussed in the context of this research effort. These criteria are consistent with the overall aim of the FEDSIM report.

For reporting purposes, we have separated the eleven criteria into two categories: general simulation criteria and DBMS-oriented criteria. The first classification is important for all simulation models, regardless of the area of simulation focus. The second portion of this chapter zeros in on criteria deemed vital to DBMS-oriented simulation facilities. These criteria will serve to localize our comparison/evaluation in the following chapter.

7.1 GENERAL SIMULATION CRITERIA

This study is focused upon three DBMS-oriented simulation facilities; nonetheless there exist several important general properties that each of these models should possess. In short, we consider 1) ease of use, 2) modeler knowledge and understanding, 3) model flexibility, 4) output facilities, 5) program product, and 6) portability. These are amply discussed and justified as important criteria in the following paragraphs.

1) Ease of Use

A simulation modeling facility is a tool; no matter how fantastic its capabilities, its value can be obtained only through use. Simple tools that achieve direct effects are often more valuable than more complex tools simply because of user preference. Thus each of the three subject simulation models will be judged on ease of use to estimate its appeal to the modeling community.

Ease of use may also entail time and cost factors. If it takes an inordinate amount of human time or computer time or storage to set up a model run, then this may overshadow the simplicity of conceputal model definition.

2) Modeler Knowledge and Understanding

This criterion is somewhat related to criterion 1, in that many difficult tasks become easier with repetition. But this criterion is meant to read like an I/O definition of a problem: how much modeler knowledge is required in order to use the simulation facility, and what understanding of the process being simulated will result due to the effect of model output on the user? A doctor may claim that a knife is very easy to use; yet much more is required to perform an appendectomy with much assurance of success. The term, modeler knowledge, thus reflects the requisite computer systems/simulation/database background required before one can use the simulation facility.

The second portion of this criterion is that of model understanding. It is not often that output statistics can be useful outside of the context of the modeled process which the statistics characterize. This criterion is not considered trivial by the author .... often the most important goal of simulation is further understanding of the process under investigation.

3) Model Flexibility

This factor addresses the long-term value of the simulation tool: how many different situations can the model address without requiring major modification? In the context of this study, the important parameters are the DBMS employed and the hardware configuration desired.

Further, we are concerned with the adaptability of the model for future concepts. Is the model's basis sufficiently general to be expandable? Is the code/design modular so that modifications/enhancements can be made without reconstructing the entire model?

Flexibility can also concern how the model can be used. Can user focus change, for instance, from throughput time to response time; from page faults to database path lengths? How flexible are the outputs generated?

Lastly we ask if the user can easily alter or expand the model to meet his needs. This is a direct reflection of how "black box" the model's design is with regard to user understanding/interaction. This facet also distinguishes between a "complete" design tool and a "completed" design tool; can I alter it to meet my needs or is it already cast in cement?

4) Output Facilities

An important attribute of any simulation model is the generated statistical output. Consideration is given to both discrete and aggregate data; how completely does it describe the simulated process? How accurate are the results? (Because of macro vs. micro modeling some data will be much more precise than other data.)

Output readability is also important; is it well-labeled? Further, is it well-organized? Can the user easily adjust it to suit his reporting needs? Is the model flexible enough to allow the user to augment its output where further statistics or traces are desired?

5) Program Product

Any design tool must be evaluated as a program product. One must examine the timeliness and the completeness of the product; is research and development continuing regardless of its stage of completeness?

Available documentation must be of sufficient clarity and detail to describe the product. External documentation should include a User's Guide and a set of documented examples of problems solved using the product. The users guide should not only describe user input specifications, but also should explain the meaning of all output statistics and yield some insight into how the input-output transformation is effected.

Internal documentation is vital if the user is to have any control over product enhancements. It further serves to document the modeling process should the user desire to understand the model better.

6) Portability

The portability of a software product relates to how easily it can be moved from one hardware system to another. Any machine idiosyncracies used by the software, such as wordsize, register usage or character encoding will hamper portability.

Languages must also be considered; any breach from a standard source language available on most computers (such as FORTRAN or COBOL) will dampen portability at this time. (In 5 years or so language should be unimportant as a portability factor, with plug-in compilers/emulators available on any machine for most source langauges.)

Lower level languages, such as Assembler or Macro-like code will always be machine-dependent and are thus undesireable. JCL (Job Control Language) must also be considered, for it differs on many computers and even different installations with the same computer. In particular heavy JCL usage by the modeler is to be avoided.

7.2 DBMS-Oriented Criteria

The general criteria defined in Section 7.1 will serve to assess the general overall value of a simulation facility. Given a satisfactory simulation facility, then one must ask, is it suitable to my specific needs? In this context we define criteria oriented towards DBMS since that serves as the focal point of our simulation study. To model a computer system, one must be able to characterize (1) hardware and (2) software. Given a modeled computer system, one cannot simulate DBMS without adequate characterizations of (3) DDL, (4) DML, and (5) DMCL. These five criteria are used collectively to comprise the DBMS-oriented criteria.

1) Hardware Characterization

Accurate and complete hardware characterization capabilities are requisite if DBMS-oriented timing analyses are requested by the modeler. As such, the facility must adequately represent the CPU execute cycle, main memory, hardware buffers, storage devices such as disc, drum, tape; card readers/punches, remote terminals, etc. Further, device connections must be specified with channel capabilities defined and controllers or whatever assigned.

For full flexibility, hardware should be able to go across vendors such as IBM, CDC, Honeywell, DEC, etc., so that any current marketed system can be simulated. Further, it should be possible to configure "mixed" systems such as IBM mainframe with Control Data discs, etc., so long as the mix is possible to construct.

Lastly, extensibility would require some way for the modeler to characterize some future device that doesn't exist today; otherwise modeling will be restricted to analyzing today's machines as opposed to also being able to answer "what if" questions about tomorrow's hardware.

2) Software Characterization

Primary to reasonable simulation of computer systems is an adequate characterization of the operating system of the host machine, if any output relating to throughput/response time is needed. Level of detail required will change depending upon exact simulation needs. Some mechanism for handling I/O must exist (e.g., an IOCS model) that can be parameterized to characterize various buffering strategies, as these have a definite effect upon important I/O considerations such as page fault rate.

The model must also provide some user mechanism for task definition so that the system workload can be defined. This may or may not include non-I/O operations as the focus is upon DBMS. However, fuller generality/specificity regarding CPU operations will serve to make the model more useful.

The DBMS is software or software/firmware itself, but is of such obvious importance that it is subdivided into its natural components in the following three sections.

3) Data Definition Language Facility (DDL)

The fundamental concepts of DBMS begin with the definitions of schemas (record definitions) and sets. The modeler must be able to define these and relate them to his task definitions that describe the system workload. Part of the DBMS simulator must then relate schema references during task execution to actual record occurrences and properly characterize the I/O and buffering required to satisfy the user task reference.

Sub-schemas should also be possible to simulate, else the database being modeled is restricted to a single schema, losing the important database advantage of logical data independence.

Set definitions must be describable as user input. The simulator must then bind set references (e.g., find next in set A) to simulated set occurrences for the DML commands disucssed in the next section.

For both schema and set, the user should be provided parameters to describe size, quantity, etc., or a capability of having an actual database to organize via schema and set would be required. Further variables, such as the use of owner or prior pointers must be modeled if available on the simulated DBMS for they affect I/O and execution times significantly.

AD-A104 930 OHIO STATE UNIV COLUMBUS DEPT OF COMPUTER AND INFORM--ETC F/G 9/2
COMPUTER SYSTEMS/DATABASE SIMULATION.(U)
OCT 78 L L ROSE

UNCLASSIFIED NL

2 OF 2
ADA
104 930

END
DATE
FILMED
10-81
DTIC

4) Data Manipulation Language Facility (DML)

Normal user task description of DBMS-related activity consists of sequences of DML commands. To force the modeler to characterize task description in any other manner entails not only unnecessary work but also unnecessary risk. To ensure integrity of task desription one should change it as little as possible from the execution task. Thus a DML facility is required of the simulator, of a form close to that used by the DBMS under scrutiny.

The simulator must be able to carry out DML commands, such as FIND NEXT or FIND OWNER, using the modeler-defined sets and attributes to effect simulation of data traversal and the resultant paging and buffering activity.

Lastly, after definition and execution, the simulator should be capable of providing a trace of the data traversal carried out: logically to show how many user records were traversed; and physically, to show actual I/O blocks read. Without this capability, user insight into DBMS activity is severely limited.

5) Data Media Control Language Facility (DMCL)

It is clear that items 3, 4, and 5 are inseparable; together they characterize the DBMS. Concerning the DMCL facility, it is here that the actual DBMS-IOCS interface is modeled. This, of course, differs with each DBMS. Hopefully, this portion of the simulator is reasonably modular and independent, so that DMCL changes to other DBMS' are not overly difficult.

The DMCL facility defines the logical-physical binding so that logical data references can be transformed into physical data references for I/O timing. Thus we see that the DML provides for the logical data request, the DDL provides the framework and direction for finding the logical data, and the DMCL transforms logical I/O requests to physical I/O requests. All three facilities are required to model DBMS effectively.

8. COMPARATIVE EVALUATION OF CASE, DIMUI, AND IPSS

Chapter 7 provides a basis for comparing and evaluating these three simulation systems. To enable comparisons to accompany evaluations, this chapter is structured around the eleven criteria cited in the previous chapter. For each criterion, all three systems will be discussed. The final section of this chapter, 8.12, provides a rough estimate of how these systems match up.

8.1 EASE OF USE

CASE, DIMUI, and IPSS are all large, complex software products. Thus it should be understood that it is a non-trivial exercise to design them to be truly easy to use. Table 8-1 shows that CASE rates better on this point than do IPSS or DIMUI; this is due to the "black box" nature of CASE more than anything else.

CASE rates "good", but it does have several shortcomings. All input is fixed-format and poorly labeled; error messages are vague and difficult to resolve; output is poorly documented; many IPA runs may be required before CPA can be executed. Task *definition*, if actual CASE commands are used, is a tedious affair. On the plus side, CASE has a large library that eases user definitions significantly, offers "general mix" instruction sets for task description that are easy to use and have proven very useful, and executing CASE runs is not particularly difficult.

DIMUI has two current shortcomings that result in a "medium" rating: stels and JCL. Task descriptions require stels to explicitly indicate currency;

the author finds this distasteful and an unnecessary exercise for the user - DIMUI should do this automatically. Executing a DIMUI run is a multi-step job with many extraneous data sets, etc.; the JCL is much more complex and verbose than it is for CASE or IPSS. DIMUI error messages are extremely poor - many commands can create the same message. Nonetheless, it is pretty easy to set-up a DIMUI run.

Preparation for an IPSS run is much more time-consuming and difficult than it is for the "black box"-oriented CASE and DIMUI systems. It is a task of another dimension ... much more a programming task as opposed to an input task. The IPSS modular design is a great aid, as are the language constructs, but model building is hard work. After the model is completed, IPSS execution is trivial (from the user standpoint) since no JCL is required; the IPSS nucleus (assembler routines) provides everything. IPSS rates low compared to CASE and DIMUI because it is a product methodology and language as opposed to a parameterized model.

8.2 MODELER KNOWLEDGE AND UNDERSTANDING

This criterion separates the three subject products into two categories: black-box (CASE and DIMUI) and creative design (IPSS). The black-box products require little understanding of the simulation process; they can be used immediately. On the output side, however, we see that you don't get something for nothing. The output from model execution lends little to no further user understanding of the simulation activity being modeled. This is advantageous where the user is non-sophisticated in simulation ... but he never will learn much from black-box models.

The IPSS facility requires a good degree of simulation background; it further requires knowledge of the IPSS language consisting of over 100 statements. However, when the model is completed, it is of the user's design. He has gained many insights into the modeled system during model construction; he can add output/trace statements wherever he desires to learn more about the system simulated. He can more effectively use this model to develop the data he requires because it is his design, not someone else's. Thus the initial effort has a good payoff in this case. IPSS forces one to understand both the system to be modeled and simulation using IPSS; the result is highly positive.

8.3 *MODEL FLEXIBILITY*

The DIMUI package ranks low on flexibility for several reasons. Hardware options are parameterized and the available library is small at present. Further, channel/device controllers, multiplexing, networking are not currently supported. The user is not told how he might add some new or fictitious hardware device to the system: hardware cache memory, bubble storage, etc.

Flexibility is also limited in DIMUI regarding any changes to output generated; you get what it offers, period. Lastly, when the DBMS changes (say IMS to IDMS) the user cannot make this change. An entire new DIMUI model (70% similar to the old one) must be used, with DIMUI experts required to carry out such a model change. This entails DDL, DML, and DMCL changes, all of which are not modularly independent in DIMUI.

CASE has somewhat better felxibility: any of 35 reports may be requested by the user. The CASE library is extensive with regard to hardware and software definitions and is a real asset to the user. Furthermore, the user is given the capability of adding or modifying (within strict definitional confines unfortunately) library entries to enhance flexibility.

Since CASE doesn't explicitly handle DDL statements, full flexibility results. Pseudo-DML commands (ignoring schema to schema translations) can be accommodated through massive IA table definitions (discussed later) but very clumsily, thus diminishing CASE's flexibility.

IPSS, by design, is a very flexible tool. All hardware-software definitions are [illegible] modules and [illegible] easily be altered. Full possibilities concerning channel connections, *etc.*, can easily be explored. The user can characterize any device imaginable, statistically or functionally in an independent, modular fashion.

IPSS output flexibility is truly excellent: any of a large number of IPSS automatic reports may be requested, plus the modeler can imbed FORTRAN code within the IPSS routines to compute, tally, store, and output whatever he desires. Changes to the DBMS under consideration will cause only modular changes to some of the IPSS code for DDL, DML, and DMCL, this was all defined by the modeler within the IPSS methodology so change should not be traumatic.

8.4 OUTPUT STATISTICS

DIMUI ranks lowest in this category due to its paucity of statistics. Output is poorly labeled, with no effort made in the DIMUI documentation to explain the significance of each statistic, or in the DIMUI code to suppress unnecessary or meaningless output (e.g., statistics on wait time for nonexistant queues).

There is little user control in DIMUI over output reports, and no way provided the user to augment DIMUI output by adding his own events/code/etc. DBMS-oriented statistics such as average path length on FIND NEXT commands, etc., are not provided; nor is it clear that they can easily be provided.

Without any of these additional datum, the DIMUI output provides the user almost no insight into what tasks or schemas, etc., are causing inefficient use of the database.

CASE ranks medium on output, for it does provide a wealth of output and user controls for report selection. No CASE documentation exists to help explain the meaning of the output with regard to relating it to changes in design/workload the user should consider.

As CASE has no concept of schemas or sets, it is literally impossible to derive good statistics relationg to database traversal. It is also unclear how one can incorporate necessary data such as average set size and deviation into CASE for simulated set traversal and reasonably accurate statistics at other than a very macro level.

IPSS ranks on top in the statistics division, simply because the user can augment the IPSS-generated statistics with his own if necessary. The DBMS portion of IPSS has implemented or proposed implementation of most features the user would request. The WAIT facility enables the user to collect WAIT statistics whenever he desires.

8.5 PROGRAM PRODUCT

None of the three systems examined can be described as a polished product. Each, however, is a usable product today, and as such the final rating of all systems was medium.

CASE is a completed, debugged system that has been available for several years. Further developmental work, to the author's knowledge, is limited to enhancements/modifications within the existing framework only. The CASE User's Guide is hardly that; it simply documents the myriad of fixed-format inputs. Little insight/discussion of how CASE works is provided. The output is poorly documented regarding any methodology of use. The source code is poorly documented (according to FEDSIM personnel) and available only on a restricted basis.

DIMUI is a research product which is yet being developed. Source code is available and well written and internally documented. The DIMUI methodology is reasonably well documented by Reiter in the user guide and supporting papers. One of its shortcomings is in documenting non-trivial examples (true of all three systems). No methodology of use has been developed for the DIMUI user, yet one would be most beneficial. Although DIMUI is a usable product today, its development into a mature, polished product continues, especially in the areas of DML translation and JCL procedures for execution-time processing.

IPSS also rates medium as a program product. Good, clear, brief documentation is sorely needed to introduce the casual user to IPSS. The language (over 100 statements) is not presented in a hierarchical fashion, so that one can begin with a small subset and construct simple IPSS models and then expand horizons. Documentation on how to use IPSS (e.g., a series of examples of varying degrees of difficulty) is nonexistant, but would be most helpful. IPSS has no library facility (like CASE, for example) for storing device and service definitions to save the user tons of work. The IPSS/DBMS model requires some finishing touches and much validation testing; it should be incorporated into the basic IPSS model to provide a comprehensive model if the user so desires. A condensed syntax and semantics document should be developed for the user.

8.6 PORTABILITY

All three of the subject systems are written in ANSI FORTRAN, to varying degrees. The assumption is that this makes the system portable over many computers. While this is not completely true, it is normally a sufficient condition to guarantee that the system will run, without undue effort, on other computers. True portability requires independence of I/O device designations, etc., which can be achieved by careful program design. Rose and Hellerman [12] discuss portability factors that can be incorporated in FORTRAN programs to enhance machine independence.

The CASE source code was not made avaialble to this author, but documentation claims that CASE is completely written in standard FORTRAN. It has been run on several different computers (IBM, Honeywell, CDC) which is a good demonstration of portability. Hence we rate, with some trepidation, good for CASE portability.

DIMUI rates fair on portability for two reasons. First, the new stage 1 procedure for DDL translation to SIDBL is written entirely in PL/1. It further requires a megabyte for execution, a seeming inordinate amount of core. DIMUI stages 2 and 3 are all FORTRAN with the exception of two tiny Assembler routines. These can easily be altered for any other machine in a trivial manner as they are only several statements long. However, they are operating on hex flags input to DIMUI, and this is clearly not at all machine-independent. Name passing to DIMUI routines (the 4H arguments) is also clearly wordsize-dependent but should not pose any great problems. Lastly, a list of JCL is required to run DIMUI and this is all installation-dependent.

The IPSS executional component is written entirely in FORTRAN which appears to be reasonably portable. However, IPSS execution is, like DIMUI, a complex procedure. Instead of placing this burden on the user (as does

DIMUI) IPSS has an Assembler nucleus which generates all necessary JCL and job steps. This must be re-written for each computer but should not pose any great difficulty. The IPSS language translator is written in PL/1, a language not supported by all computers.

Both IPSS and DIMUI could probably re-write their parsers into FORTRAN with the aid of routines such as BYTRAN [13] which stress portability by parameterizing factors such as collating sequence, device names, wordsize, etc. However, the programs would no doubt be larger and less efficient; it is questionable whether it is a worthwhile endeavor. FORTRAN has recently made in-roads on character processing and, in this author's opinion, will soon incorporate language features for character processing. When this is standardized, then re-writes using FORTRAN would be in order.

8.7 HARDWARE CHARACTERIZATION

All three systems appear to have adequate mechansims for hardware characterization. CASE has a full library which the user simply references; all legal device inter-connections are permitted. Normal maintenance by TESDATA keeps the library up-to-date. Timing tables are unique by vendor and quite sufficient.

DIMUI has a reasonable characterization of the CPU, disc, and tape and single-control channels, but little else. This tiny DIMUI library requires significant expansion. Networking and multiplexing may impose severe demands on the stage 3 model design, but should be considered. A new stage 3 can always be constructed should further sophistication be required; it may be extensible within the current model design.

IPSS flexibly allows the user to characterize devices, etc., either by static tables or dynamically by function definitions. The only thing lacking is some help for the user; a hardware library which the user could simply reference would be highly desirable. In IPSS one can define any hypothetical hardware device; in DIMUI this cannot be done; in CASE it can be done only within the confines of the device definition structural parameters.

8.8 SOFTWARE CHARACTERIZATION

We consider two major software components: characterization of the operating system, and characterization of user tasks/programs. Emphasis, for evaluative purposes, was placed on the latter since our focal point is database and I/O activity rather than CPU time.

CASE has a reasonably good model of the operating system, although it is not event-oriented. Interrupts are not handled explicitly, as we saw in the CASE overview chapter. One poor exception to CASE modeling is that overlapped I/O on the same device is assumed random, and timed accordingly. In some cases (e.g., 1 access file A, 1000 sequential accesses file B) this can generate misleading results (1001 random accesses instead of 2 random and 999 sequential accesses).

CASE task representation is excellent (excepting DML's which we discuss shortly). All general or specific processing (CPU and file) can be described with great accuracy using CASE language constructs. Hence workload characterization can be accomplished very nicely in CASE, to include scheduling constraints, SORTS, reporting requirements, etc.

DIMUI doesn't rate as well because of its capabilities for characterizing user workload. This is due to DIMUI's emphasis on I/O time rather than CPU time. the function CPUTIM(n) must be invoked by the user to increment the clock by n units to simulate CPU-bound processing. This is extremely primitive compared to CASE.

The author cannot comment in great detail about DIMUI O/S modeling, for the necessary document was unavailable and the technical advisor, Mr. Isaac Bussel, was not an expert on stage 3. The design, at least, of the O/S to model all important activities relating to I/O appears reasonable.

IPSS provides an excellent environment for the characterization of software. User programs can be written, and are executed, to characterize processing with probabilistic looping, etc., much like the CASE design. The operating system is just another IPSS process, and can be modeled at any desired level of detail. It would be helpful to relieve the user of the burden of O/S definition. The FEDSIM report also points out some further facilities that should be included automatically in IPSS. Writing a detailed O/S in IPSS is non-trivial.

8.9 DATA DEFINITION LANGUAGE FACILITY

As Table 8-1 illustrates, CASE has no current facility for DDL characterization. CASE was not initially designed for DBMS activity, and adding schema and set structures is almost impossible unless the CASE specifications are significantly changed. CASE has no mechanism even for record naming; it simply references files which contain records of a given type. Hence all database references (see next two sections) require implicit knowledge of a fixed schema.

IPSS and DIMUI both provide the user normal statements for record schema and set definitions. For the simulation to be viable and useful, this author feels it mandatory that the user have this explicit facility.

DIMUI has two current restrictions that IPSS does not impose. First, set members must all be of the same record type. Efforts are now underway to resolve this restriction. Second, and of much greater significance, is the fact that DIMUI does not support subschemas. I.e., the schema is fixed, whereas IPSS allows an arbitrary number of subschemas through schema to schema transformation at each level of the system.

8.10 DATA MANIPULATION LANGUAGE FACILITY

The CASE DML facility is really at the level below logical data manipulation, but it can be used to characterize database tasks. Since there are no schema definitions in CASE the user must rely upon the record definitions - i.e., the physical layout of the database. Using the IA tables (defined in the next section) of CASE one must refer to the IA table that should correspond to the simulated DML command. For example, a FIND OWNER command is simulated in CASE by a GET (read) command associated with a particular IA table which describes the access path and actual number of disc accesses required to perform the operation. Hence, for all different possiblities (owner pointer, via, immediate owner, etc.) the CASE task definer must mix his GETs and IA tables to simulate database traversal. If it is done correctly, then reasonable I/O timings can result. Path analysis, of course, is impossible since no record of record paths is possible in CASE. Further, any change in the DBMS implemented will alter the CASE task description.

Both IPSS and DIMUI provide a normal DML capability for database task description. The IPSS DML statements correspond directly to IDMS commands; DIMUI DML statements are similar but not identical and require "stel" references. If the user is not careful to drop DIMUI stels when they are no longer required, significant storage can accrue and cause problems.

Database traversal is at the logical record level in both systems, but DIMUI does not provide "hooks" as does IPSS, so that statistics such as logical path length can be obtained. These are extremely useful for modeler understanding of schema efficiency, set definitions and storages, etc., and should be included. The fact that DIMUI does actual traversal in SIDBL should not inhibit its capability to create these data trails and statistics concerning their use.

8.11 DEVICE MEDIA CONTROL LANGUAGE FACILITY

A DMCL facility is necessary to enable the logical records in the database to be bound to physical records. At the DMCL level we see the actual DBMS physical structure. In both IPSS and DIMUI we find a vehicle for database physical structure definition, albeit actual code in both instances. However, once we have the DBMS implementation schema defined, we can use it to simulate database traversal to effect I/O requests with appropriate buffering strategies.

In the CASE system, the Immediate Access (I/A) Tables are the natural place to characterize DBMS physical implementation. CASE provides three such tables, to describe sequential, indexed-sequential, and random processing. FEDSIM, under contract to AIRMICS (project NA-107-007-ARMY) has defined four more I/A tables, to denote, under IDMS, these four activities: 1) CALC access to owner/member; 2) DIRECT access to owner/member; 3) VIA access by member with CALC owner; 4) VIA access by member with DIRECT owner. While these tables will reflect average DBMS processing to some degree of accuracy, many other DBMS activities can occur (do owner/prior pointers exist, etc.) and each of these requires another I/A table. FEDSIM simulated the TOTAL DBMS using 64 I/A tables, and this author believes TOTAL to be less complex than IDMS. The user must redefine the same file for all of its associated I/A tables, and must reference it with the appropriate table to denote accessing during task definition. Thus we find that I/A tables provide a mechanism for predicting average access time to traverse a node in the database (in conjunction with the database definition cards which handle logical-physical segments and buffers), but are extremely clumsy and suspect to abuse and misuse leading to clear validation problems.

8.12 TABULAR COMPARISON

Table 8-1 illustrates a gross overview of the preceding discussions in this chapter. It shows, in a macro-level evaluation, how these three systems stack up against each other.

One can derive a single Figure of Merit for each of these systems by merely assigning weights to the eleven criteria and providing values, such as 1, 2, and 3 to the scores low, medium, high.[1] This figure of merit is quite suspect when utilized without any other data, but provides a good summarization of Table 8-1.

This author suggests assigning a weight of 2 (meaning most important) to the DBS-oriented criteria: 7 (hardware), 8 (software), 9 (DDL), 10 (DML), and 11 (DMCL). To the other 5 criteria are assigned the lesser weight of 1. Given these assignments we have the following results for F(sys), our Figure of Merit for each system:

$$F(\text{case}) = 3 + \frac{(3+1)}{2} + 2+2+2+3+6+6+0+2+2 = 30$$

$$F(\text{dimui}) = 2 + \frac{(3+1)}{2} + 1+1+2+2+6+4+6+6+6 = 38$$

$$F(\text{ipss}) = 1 + \frac{(1+3)}{2} + 3+3+2+2+6+6+6+6+6 = 43$$

[1]For the user knowledge criterion, which shows two scores, the first score is valued inversely (e.g., high = 1, and then averaged with the second value.

		CASE	DIMUI	IPSS
GENERAL	1. Ease of Use	High	Medium	Low
	2. User Knowledge	Low	Low	High
	3. Model Flexibility	Medium	Low	High
	4. Output Statistics	Medium	Low	High
	5. Program Product	Medium	Medium	Medium
	6. Portability	High	Medium	Medium
DBMS ORIENTED	7. Hardware	High	High	High
	8. Software	High	Medium	High
	9. DDL Facility	Zero	High	High
	10. DML Facility	Low	High	High
	11. DMCL Facility	Low	High	High

TABLE 8-1. The Three Systems Contrasted

A quick study of our scoring system shows that the maximum score for F(sys) = 48. It is perhaps more appropriate to consider the "relative" scores, which we denote $\hat{F}(sys)$:

$\hat{F}(case) = 30/48 = .625$

$\hat{F}(dimui) = 38/48 = .792$

$\hat{F}(ipss) = 43/48 = .896.$

The function $\hat{F}$ shows that CASE meets only 60% of our DBMS modeling needs, that DIMUI is at a much more satisfactory 80% level, with IPSS slightly better at near the 90% level.

The author's estimate is that future product development of a moderate nature could extend CASE's value by 5% to around 67.5%; lack of a true database focus limits its capability to easily incorporate the needed DDL, DML, and DMCL concepts.

Similarly, DIMUI can be ehanced in the output and model flexibility areas by 5% to extend its figure of merit to near the 85% mark. Moderate future enhancements to IPSS should increase its utility and marketability, but the Figure of Merit would not rise much above the 90% level, a difficult limit to exceed.

9. RECOMMENDATIONS

This report has overviewed database modeling constructs of three simulation systems: CASE, DIMUI, and IPSS. In this final chapter the author makes his recommendations as to how he believes AIRMICS can best use these simulation systems, and what areas he considers worthwhile for further developmental efforts. These opinions are strictly those of the author, and do not necessarily reflect the opinions of personnel at Ohio State University, AIRMICS, or FEDSIM. Each of the three systems is considered separately in the following chapter subsections.

9.1 CASE USAGE AND FURTHER DEVELOPMENT

The CASE system, as it currently exists, is an easy to use, cheap, useful process-oriented computer systems analyzer. Its main use at USACSC has been in a batch and file-oriented environment; in this role it appears to meet USACSC needs within reason. However, when we consider using CASE to do DBMS analysis, we're considering the wrong kind of tool. The addition of new I/A tables and Database Definition Cards may enable a near-expert to achieve a reasonable simulation facsimile of the desired system, but it doesn't come close to providing an acceptable tool for database analysis. This is because I/O output isn't enough to tell the modeler how his database is being utilized. Furthermore, without any user schema or set definition capability, validation is an arduous exercise at best.

As a hybrid model for DBMS analysis, CASE just cannot ever provide data about events or structures not known to the model. This author believes that any future DBMS simulator should be designed to provide the broadest possible types of outputs. CASE is limited, since it was not initially designed with database management in mind, to I/O timing outputs, and these represent the tip of the iceberg only to the modeler. Thus further expenditures on CASE for modeling database are not warranted unless the basic CASE design for modeling data reference and flow is altered, a gigantic task it would seem.

Further work with CASE would prove useful in the following three areas:

1) FEDSIM to include CASE in its report [11] along with IPSS, DIMUI, and ECSS. This would help strengthen AIRMICS' comparison of these three systems and further document CASE capabilities.
2) FEDSIM to prioritize the over 200 concepts in its report [11] so that the presence or absence of any given concept can be properly evaluated within the database context. In particular, this would serve to enable one to devise a linear objective function for further simulation system contrasts and would demonstrate the importance of those database constructs lacking in CASE.
3) The CASE User's Guide can be enhanced greatly to go beyond its current inclination to simply show the format of each input record. This would help serve all CASE users, and hopefully help the user understand the model he has defined for CASE. Any "black box" simulator must be clearly explained else it cannot be used to its fullest potential. I don't believe that CASE is being effectively used by CSC personnel due to deficiencies in both TESDATA training (no actual CASE models are run in the week-long course) and CASE documentation.

9.2 DIMUI USAGE AND FURTHER DEVELOPMENT

DIMUI has not been used much by AIRMICS or anyone else, since it is a newly developed system. However, DIMUI is an easy to use, useful database analysis tool as was demonstrated in August 1978 to USACSC and AIRMICS personnel. While DIMUI is not quite fully completed, it does represent a tool that was designed with database analysis in mind. As a "black box" simulator it is similar to CASE; yet it can simulate data structure and flow which CASE cannot.

This author recommends that DIMUI be further enhanced so that its effectiveness as a database analysis tool can be optimized. Given its basic design, most enhancements should be able to be accommodated without requiring fundamental changes to the existing DIMUI foundational methodology. There are only two exceptions to this:

1) First is the single schema design of DIMUI - subschema characterization would expand DIMUI's generality significantly; yet I believe this concept will be difficult to imbed in DIMUI - at such an expense as to perhaps not be worthwhile.

2) Second is the lack of process-oriented facilities (such as CASE employs so superbly) to model CPU activity. CASE can't adequately model obtaining a record; DIMUI can't adequately model program instruction processing given the probable record contents. A DIMUI design assumption of I/O-bounded systems assured macro-level only modeling of all I/O-oriented CPU activities. As was true for exception 1), this too may not be a cost-effective developmental activity for DIMUI.

Fertile areas do exist, nonetheless, for great enhancements to DIMUI within its basic design at reasonable costs of time and effort. Some of these are noted below. Others can no doubt be elicited from the model's creator: Allen Reiter.

1) First and foremost, DIMUI output should be enhanced. Little emphasis has been placed here, yet the output is critical to the user. Labeling of existing output data is poor and sometimes misleading. Output statistics that relate to logical record traversal are nonexistant; neither are any set statistics issued.

2) The stage 1 task description facility could be altered so that user input looks more like normal DML statements. Stels are clumsy although perhaps not easy to remove from user explicit code; even more important to remove is the user requirement to issue an artificial call to ESTSET. Clearly it would be far preferable for ease of use, to have both stels and ESTSET appear implicitly rather than explicitly in the user task description.

3) Considering portability, DIMUI uses hex flags and 4 character record names - both clear references to a 32 bit 2's complement machine which impedes portability considerably. These can be altered to FORTRAN without undue labor, albeit clumsy. Of course, the stage 1 schema preprocessor, written in PL/1, is not easy to code in FORTRAN, but it is possible.

4) FEDSIM should be charged, given further DIMUI documentation as exists today (with Reiter's DIMUI technical reports for background) to complete its evaluation of DIMUI.

5) DIMUI, to be a truly viable database tool, must have an accompanying methodology of use. If it is to be an effective tool for database tuning, then directions must be available. Further examples, etc.. may also serve to demonstrate the breadth of problems in database design, usage, and performance that are solvable using DIMUI.

9.3 IPSS USAGE AND FURTHER USAGE

IPSS, like DIMUI, is a recently developed system, and has had few users outside the domain of the team of researchers who worked to make the IPSS design methodology a reality. It was demonstrated effectively in summer 1977 to AIRMICS that IPSS could (as did DIMUI) model a backend DBMS. Recent support by the Social Security Administration to utilize IPSS as its publica-language should serve to enhance its image and its capabilities as further research and development with IPSS continues.

IPSS is a language, and hence, much more flexible (and prone to misuse) than CASE or DIMUI. It provides the modeler the tools to design any computer/database system he so desires, at whatever level of detail he deems appropriate. As such, the potential for IPSS in computer database analysis is much more extensive than other fully designed models. At the same time, IPSS usage is a more time-consuming, costly matter.

Much research should continue on IPSS, mainly because its design appears sound yet extremely flexible, and no other language like it currently exists for computer/database simulation. Some areas that should be emphasized to reap beneficial results are as follows:.

1) IPSS/DBS has not been completely tested and validated; this should be done immediately. Statistics recommended for database analysis should be implemented if not already within IPSS/DBS.
2) IPSS and IPSS/DBS should be integrated so that the entire system (with the capabilities of CASE and DIMUI together) can be utilized.
3) An IPSS library facility should be constructed and maintained to ease user definition of the DBMS, hardware devices, etc. This would help reduce the modeling effort significantly.

4) Much more concise documentation is required so that one can "ease into" IPSS without feeling lost. As IPSS now has well over 100 statements, these should be modularized hierarchically so that the user can first build a valid high-level IPSS model. Then when the user desires more microscopic modeling of any facet of the simulation, he can delve more deeply into the manuals. Further directions for model-building would be extremely helpful - for example, noting the essential ingredients for each of the six IPSS modules.

5) Any large system like DIMUI or IPSS, even written completely in FORTRAN, is not truly easy to transport to different architectures. Much of the IPSS language translation is written in PL/1 and the IPSS nucleus is Assembler as noted earlier. Translation into complete FORTRAN is impossible if the user is to be afforded the nice JCL advantages of IPSS.

9.4 SUMMARY

This chapter has shown that the author has found all three simulation systems (CASE, DIMUI, and IPSS) to be of considerable value for simulating computer systems. None are recommended to be discarded, although CASE is not a good prospect for database enhancements. DIMUI and IPSS are different, perhaps even complementary tools that are both valuable in their own right. DIMUI is a simpler system, in design, implementation, and usage - its focus lies strictly in database analysis. IPSS is much more general, much broader in scope; as a language it can be used to create models of varying detail and focus; it is a powerful, complex tool. Both DIMUI and IPSS are viable tools today for database analysis; both can be enhanced to be even more satisfactory simulation tools.

REFERENCES

1. Shannon, R. E., Systems Simulation, the Art and Science, Prentice Hall, Englewood Cliffs, New Jersey, 1975, p. 2.

2. Efron, R. and Gordon, G., "A General Purpose Digital Simulator and Examples of its Application: Part 1 - Description of the Simulator", IBM Systems Journal, III, No. 1 (1964), 21-34.

3. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", CACM (13), pp. 377-387, 1970.

4. Hauk, E. J., "Hierarchical Data Files Cut Storage Requirements", Data Processing, May 1971.

5. CODASYL Data Base Task Group Report, available from the Associating for computing Machinery (ACM), New York, April 1971.

6. CASE User Manual, produced by TESDATA Systems Corporation.

7. Reiter, Allen, "DIMUI-IDMS User Manual, Version 1.3", TR 133, TECHNION-Israel Institute of Technology, August 1978.

8. Reiter, Allen, "On Performance Modelling of Database Management Systems - An Inductive Approach", TR 1648, University of Wisconsin, Mathematics Research Center, July 1976.

9. DeLutis, T. G., "A Methodology for the Performance Evaluation of Information Processing Systems", Final Report to National Science Foundation, OSIS, GN 36622, 1977.

10. DeLutis, T. G., "Information Processing System Simulator (IPSS): Syntax and Semantics", Volumes 1 and 2, working documents to NSF Grant 36622, 1978.

11. "Evaluation of DBMS Modeling Approaches", FEDSIM Report MV-027-033-ARMY, February 1978, Washington, D. C.

12. Rose, L. L. and Hellerman, H., "Portable Character Processing in FORTRAN and Fixed Integer Environments", IEEE Transactions on Software Engineering, Vol. SE-2, #3, Sdptember '76, pp. 176-185.